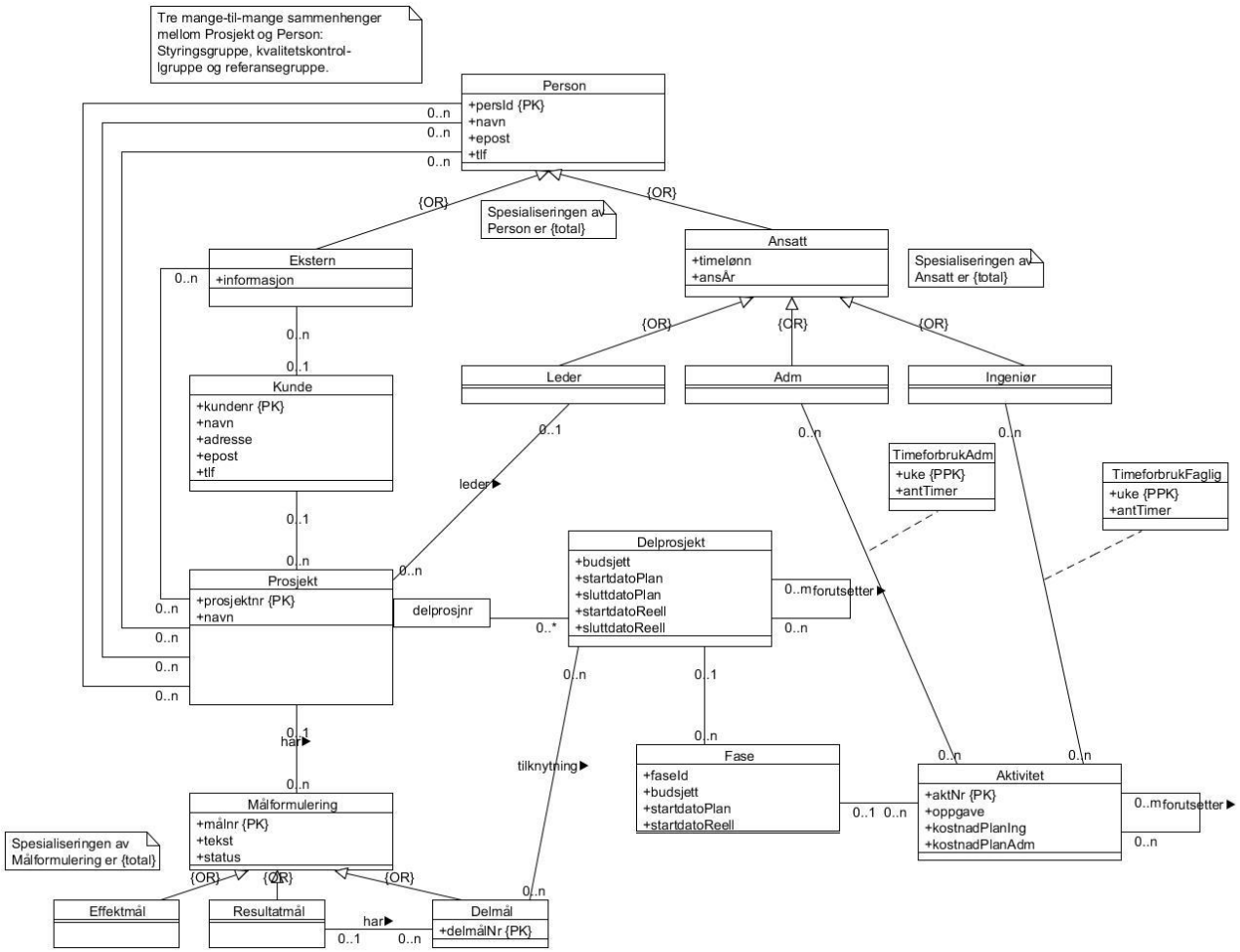
TDAT1005 Databaser med videregående programmering

Eksamen 19.mai 2017 – Løsningsforslag

OPPGAVE 1A

EER-modell



Det er ikke trukket dersom man bruker «type» e.l. i stedet for spesialisering av ansatt. Man mister imidlertid da særskilte timelister for adm og ingeniør. Også andre elementer i oppgaven har løsninger som er likeverdige med det som er vist her.

Relasjonsmodell:

person(<u>persId</u>, navn, epost, tlf)
prosjekt(<u>prosjektnr</u>, navn)
referansegruppe(<u>persId*, prosjektnr*</u>)
kvalitetskontrollgruppe(<u>persId*, prosjektnr*</u>)
styringsgruppe(<u>persId*, prosjektnr*</u>)
ekstern(<u>eksternPersId</u>*, informasjon, kundenr*)
ansatt(<u>ansattPersId</u>*, timelønn, ansÅr)
leder(<u>lederAnsattPersId</u>*)
adm(<u>admAnsattPersId</u>*)
ingeniør(<u>ingAnsattPersId</u>*)

kunde(kundenr, navn, adresse, epost, tlf)
prosjekt(prosjektnr, navn, lederAnsNr*, kundenr*))
målformulering(målnr, tekst, status)
effektmål(effektMålnr*)
resultatmål(resultatMålnr*)
delmål(resultatDelmålNr*, resultatMålnr*)
delprosjekt(prosjektnr*, delprosjnr, bnudsjett, startdatoPlan, sluttdatoPlan, startdatoReell, sluttdatoReell)
måltilknytning(resultatDelmålNr*, (prosjektnr, delprosjnr)*)
avhengigheterDelprosjekt((prosjektnr, delprosjnr)*, (forutsetterProsjektnr, forutsetterDelprosjnr)*)
fase(faseId, budsjett, startdatoPlan, startdatoReell, (prosjektnr, delprosjnr)*)
aktivitet(aktNr, oppgave, kostnadIngPlan, kostnadIngAdm, faseId*)
avhengigheter(aktNr*, forutsetterAktNr*)
timeforbrukAdm(admAnsattPersId*, aktNr*, ukenr*, antTimer)
timeforbrukFaglig(ingAnsattPersId*, aktNr*, ukenr*, antTimer)

OPPGAVE 1B

i) Finn ansattnummeret til alle ansatte som både har fullført et kurs og vært kursholder.

```
SELECT DISTINCT KURSKJØRING.kursholder FROM KURSKJØRING,DELTAKELSE
WHERE (KURSKJØRING.kursholder=DELTAKELSE.ans_nr AND
DELTAKELSE.fullført=TRUE);
```

Alternativt:

```
SELECT DISTINCT ans_nr FROM DELTAKELSE
where fullført=true AND ans_nr IN
(SELECT kursholder FROM KURSKJØRING);
```

Noen inkluderer også en test på dato i KURSKJØRING for å sikre seg at man ikke får med ansatte som er satt opp som kursholdere i kurs som ennå ikke er kjørt. Dette godt tenkt, men løsningene over holder for å få full uttelling.

ii) Skriv ut antall forskjellige sertifiseringer som bedriften tilbyr, og antall kurs som ikke tilbyr en sertifisering.

```
SELECT COUNT(DISTINCT sert_nr), COUNT(*)-COUNT(sert_nr) FROM KURS;
```

iii) Skriv ut navnet på alle sertifiseringer og hvor mange ansatte som har tatt hver av dem, sortert i synkende rekkefølge.

```
SELECT SERTIFISERING.sert_navn 'sertifisering', COUNT(ans_nr) 'antall'
FROM KOMPETANSE, SERTIFISERING
WHERE KOMPETANSE.sert_nr=SERTIFISERING.sert_nr
GROUP BY SERTIFISERING.sert_nr
ORDER BY COUNT(ans_nr) DESC;
```

iv) Skriv ut ansattnummeret til eventuelle ansatte som har fullført alle kurs i KURS.

```
SELECT ans_nr FROM DELTAKELSE
WHERE fullført=TRUE
GROUP BY ans_nr
```

```
HAVING COUNT(DISTINCT kurs_nr) IN
(SELECT COUNT(*) FROM KURS);
```

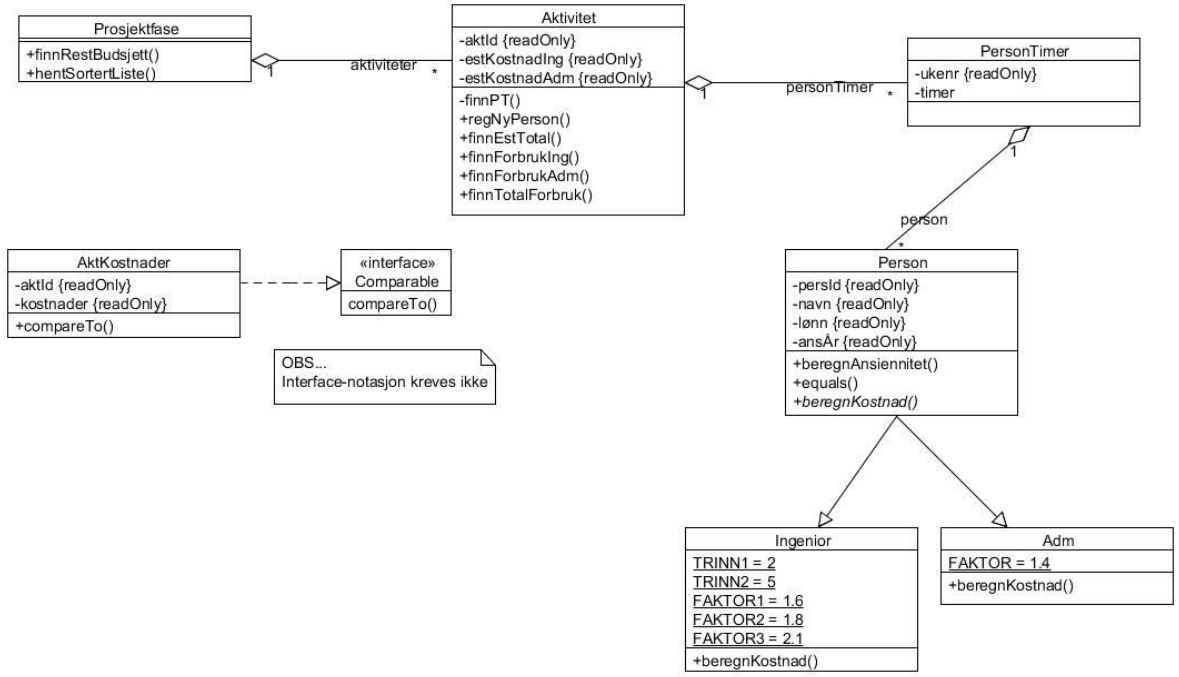
v) Finn navnet på alle sertifiseringer som minst en ansatt har men som bedriften ikke tilbyr.

```
SELECT DISTINCT sert_navn, sert_nr FROM SERTIFISERING NATURAL JOIN KOMPETANSE
WHERE sert_nr NOT IN
(SELECT sert_nr FROM KURS WHERE sert_nr IS NOT NULL);
```

Alternativ:

```
SELECT sert_navn FROM SERTIFISERING
WHERE sert_nr NOT IN (SELECT sert_nr FROM KURS WHERE sert_nr IS NOT NULL)
AND sert_nr IN (SELECT sert_nr FROM KOMPETANSE);
```

OPPGAVE 2A



OPPGAVE 2B

Vektlegger bl.a følgende punkter:

- Klassestrukturen
- Konstruktører
- Polymorfien
- Protected skal ikke brukes på variabler, i stedet skal man bruke get-metoder i subklasser
- Navngiving av konstanter
- Exceptionshåndtering i konstruktører (og set-metoder) er ikke tatt med i løsningsforslaget, men gir pluss på den måten at det kan veie opp for andre mangler – gjelder hele oppgave 2.

```
abstract class Person {
  private final int persId;
  private final String navn;
  private final int lønn;
  private final int ansÅr;

  protected Person (int persId, String navn, int lønn, int ansÅr) {
    this.persId = persId; // ikke spurt om i oppgaven
    this.navn = navn; // ikke spurt om i oppgaven
    this.lønn = lønn;
    this.ansÅr = ansÅr;
  }
  public int getPersId() {
    return persId;
  }
  public String getNavn() {
    return navn;
  }
  public int getLønn() {
    return lønn;
  }
  public int getAnsÅr() {
    return ansÅr;
  }
  public int beregnAnsiennitet() {
    return 2017 - ansÅr; // kan gjøres generell
  }

  abstract public double beregnKostnad(int antTimer);
}

class Ingenior extends Person {
  private final static int TRINN1 = 2; // 0-2 år
  private final static int TRINN2 = 5; // 3-5 år
  private final static double FAKTOR1 = 1.6;
  private final static double FAKTOR2 = 1.8;
  private final static double FAKTOR3 = 2.1;
  public Ingenior (int persId, String navn, int lønn, int ansÅr) {
    super(persId, navn, lønn, ansÅr);
  }

  @Override
  public double beregnKostnad(int antTimer) {
    int ansiennitet = beregnAnsiennitet();
    if (ansiennitet > TRINN2) {
      return getLønn() * FAKTOR3 * antTimer;
```

```
    } else if (ansiennitet > TRINN1) {
      return getLønn() * FAKTOR2 * antTimer;
    } else {
      return getLønn() * FAKTOR1 * antTimer;
    }
  }
}

class Adm extends Person {
  private final static double FAKTOR = 1.4;
  public Adm (int persId, String navn, int lønn, int ansÅr) {
    super(persId, navn, lønn, ansÅr);
  }

  @Override
  public double beregnKostnad(int antTimer) {
    return getLønn() * FAKTOR * antTimer;
  }
}
```

OPPGAVE 2C

Noen har lagt timeforbruket inn i personobjekter, for at det skal bli riktig må man ha egne personobjekter for hver aktivitet. Ellers blir alle timene liggende i samme objekt.

Bruk av tabeller med hardkodet lenge på 52 (for antall uker) trekker litt. Løsningen bør være generell slik at antall uker kan være både mer og mindre enn 52.

For øvrig kan andre løsninger enn den foreslåtte også gi full uttelling. Den foreslåtte løsningen med klassen PersonTimer baserer seg på mange-til-mange-sammenhengen mellom aktivitet og person (og uke), slik man har i datamodellen. Om man ønsker «ekte» mange-til-mange kan man ta inn this (aktivitet) som objektvariabel i tillegg til person.

Merk at equals() må være programmert dersom kandidaten har valgt en løsning der equals() mellom objekter av egendefinerte klasser benyttes, f.eks. indexof() i en arraylist.

Vi trenger equals() i klassen Person:

```
public boolean equals(Object obj) {  // klassen Person
  if (!(obj instanceof Person)) {
    return false;
  }
  if (obj == this) {
    return true;
  }
  Person p = (Person) obj;
  return (persId == p.persId);
}
```

```
class Aktivitet {
  private final int aktId;
  private final int estKostnadIng;
  private final int estKostnadAdm;
  private java.util.ArrayList<PersonTimer> personTimer =
                                  new java.util.ArrayList<PersonTimer>();
  protected Aktivitet(int aktId, int estKostnadIng, int estKostnadAdm) {
    this.aktId = aktId;
    this.estKostnadIng = estKostnadIng;
    this.estKostnadAdm = estKostnadAdm;
  }
  public int getAktId() {
    return aktId;
  }
  public int getEstKostnadIng() {
    return estKostnadIng;
  }
  public int getEstKostnadAdm() {
    return estKostnadAdm;
  }

  private int finnPT(Person p, int ukenr) {  // Hjelpemetode til bruk nedenfor
    for (int i = 0; i < personTimer.size(); i++) {
      PersonTimer pt = (PersonTimer) personTimer.get(i);
      if (pt.getPerson().equals(p) && pt.getUkenr() == ukenr) {
        return i;
      }
    }
    return -1;
  }

  public void regNyPersonTimer(Person p, int ukenr, int timer) {
    int index = finnPT(p, ukenr);
    if (index >= 0) {
      PersonTimer pt = new PersonTimer(p, ukenr, timer);
      personTimer.add(pt);
    } else {
      int t = personTimer.get(index).getTimer();
      personTimer.get(index).setTimer(t + timer);
    }
  }
}

class PersonTimer {
  private final Person p;
  private final int ukenr; // ingen datakontroll
  private int timer;  // ingen datakontroll
  public PersonTimer(Person p, int ukenr, int timer) {
```

```
    this.p = p;
    this.ukenr = ukenr;
    this.timer = timer;
  }
  public Person getPerson() {
    return p;
  }
  public int getUkenr() {
    return ukenr;
  }
  public int getTimer() {
    return timer;
  }
  public void setTimer(int timer) {
    this.timer = timer;
  }
}
```

OPPGAVE 2D

Noen hadde akkumulert kostnadene i egne objektvariabler i aktivitetsobjektene. Dette er ikke god programmeringsskikk. Disse verdiene kan enkelt regnes ut, som vist nedenfor.

Klassen Aktivitet:

```
  public double finnForbrukIng() {
    double sum = 0.0;
    for (PersonTimer pt : personTimer) {
      Person p = pt.getPerson();
      if (p instanceof Ingenior) {
        sum += p.beregnKostnad(pt.getTimer());
      }
    }
    return sum;
  }

  public double finnForbrukAdm() {
    double sum = 0.0;
    for (PersonTimer pt : personTimer) {
      Person p = pt.getPerson();
      if (p instanceof Adm) {
        sum += p.beregnKostnad(pt.getTimer());
      }
    }
    return sum;
  }
```

OPPGAVE 2E

Klassen Prosjektfase:

```
public double finnRestBudsjett() {
  double sum = 0.0;
  for (Aktivitet akt : aktiviteter) {
    sum += (akt.finnEstTotal() - akt.finnTotalForbruk());
  }
  return sum;
}
```

Flere metoder i klassen Aktivitet:

```
public int finnEstTotal() {
  return estKostnadIng + estKostnadAdm;
}

public double finnTotalForbruk() {
  return finnForbrukAdm() + finnForbrukIng();
}
```

OPPGAVE 2F

Her er det tre ting å merke seg:

Resultatet skal være en tabell, ikke en ArrayList (eller annen Java API-liste). I tillegg til at det står i klartekst, så bør også beskjeden om at java.util.Arrays.sort() (og ikke Collections.sort()) skal brukes gi et hint om dette. Imidlertid er ordet «liste» brukt flere ganger i oppgaveteksten, det er derfor ikke trukket mye akkurat på dette punktet.

Elementene i tabellen skal inneholde aktivitetsidentifikasjon og totale kostnader. For å få til det lager man en klasse med disse to som medlemmer.

Å bruke en sorteringsmetode fra Java API-et krever enten at interfacet Comparable er implementert for det som skal sorteres, eller at det lages en egen Comparator-klasse.

Ny metode i klassen Prosjektfase:

```
  public AktKostnader[] hentSortertListe() {
    AktKostnader[] liste = new AktKostnader[aktiviteter.size()];
    for (int i = 0; i < liste.length; i++) {
      Aktivitet akt = aktiviteter.get(i);
      liste[i] = new AktKostnader(akt.getAktId(), akt.finnTotalForbruk());
    }
    java.util.Arrays.sort(liste);
    return liste;
  }
}
```

Ny klasse:

```
class AktKostnader implements Comparable<AktKostnader> {
  private int aktId;
  private double kostnader;
  public AktKostnader(int aktId, double kostnader) {
    this.aktId = aktId;
    this.kostnader = kostnader;
  }
  public int getAktId() {
    return aktId;
  }
  public double getKostnader() {
    return kostnader;
  }
  public int compareTo(AktKostnader denAndre) {
    final double TOLERANSE = 0.001;
    double kost1 = kostnader;
    double kost2 = denAndre.getKostnader();
    if (Math.abs(kost1 - kost2) < TOLERANSE) {
      return 0;
    } else if (kost1 < kost2) {
      return -1;
    } else {
      return 1;
    }
  }
}
```

OPPGAVE 3A

i: metode 1

ii: kompileringsfeil

iii: metode 1BD

iv: metode 1BD

OPPGAVE 3B

For at programmet skal kunne reagere på brukerens manipulering av GUI-objekter (kilder), må disse komponentene ha et lytterobjekt knyttet til seg som reagerer på relevante brukerhandlinger. Lytteren implementerer et relevant interface (f.eks. ActionListener for knapper) for å kunne motta standardiserte meldinger og registreres hos Java-tolkeren. Selve knappetrykket kalles en hendelse. Det som da skjer hvis en bruker trykker på en knapp er:

- OSet mottar signaler fra fysiske I/O-enheter, og tolker disse til for eksempel et knappetrykk med en mus. Den ser at knappen tilhører et Java-program og varsler Java-tolkeren om at en bruker har trykket på knappen.

- Tolkeren genererer så en actionPerformed()-melding (del av ActionListener-interfacet) og sender denne til knappens lytter.
- Lytteren og derigjennom programmet kan reagere på meldingen som den vil gjennom sin implementasjon av actionPerformed().

OPPGAVE 3C

Tapt oppdatering er når to transaksjoner leser samme verdi og begge oppdaterer uavhengig av hverandre slik at den første transaksjonens oppdatering blir overskrevet av den andre.

Ikke-overgitte data er når en transaksjon får tilgang til data fra en annen før disse er endelige, og dermed ikke nødvendigvis i en riktig tilstand. Den første transaksjonen kan for eksempel velge å avbryte slik at den andre har lest ikke-gyldige data.

Inkonsistente innhentinger er når en transaksjon leser data som bare er delvis oppdatert av en annen transaksjon. Hvis den handler ut fra dette kan vi få inkonsistens i databasen.