



Avdeling for allmenne fag

EKSAMEN

852 OBJEKTORIENTERT PROGRAMMERING

07.06.2004

Tid:	9.00 – 13.00
Målform:	Nynorsk/bokmål
Sidetal:	forside + 3 + 3
Hjelpemiddel:	Alle trykte og skrevne
Merknader:	Ingen
Vedlegg:	Ingen

Eksamensresultata vert offentleggjort på Arena høgskole. Passordet til Arena har alle studentar fått i eige brev ved semesterstart. Du går inn på Arena høgskole gjennom HiTs nettsider.

Eksamen 852 Objektorientert Programmering 07.06.2004 - Nynorsk

Råd og retningslinjer. Les oppgåveteksten godt før du går i gang med å løyse oppgåva. Deloppgåvene er uavhengige av kvarandre i den meining at om du ikkje får til éi oppgåve, kan du likevel gjere neste, som om den fyrste var løyst. Fordél tida godt på alle oppgåvene. Om du meiner ei oppgåve er upresis, så skriv din eigen presisering.

Oppgåve 1 (35%)

Du skal utvikle programvare for eit telefonselskap. Selskapet har eit antal kundar. Kvar kunde har null eller fleire telefonabonnement. Eit telefonabonnement er enten fast-telefon eller mobiltelefon. Fast-telefonen er enten av type "analog" eller "ISDN".

Mobiltelefonabonnementet er enten type "kontantkort", "ungdom", "privat" eller "privat pluss", desse tilbyr litt ulike tenester og har litt ulike prisar på dei ulike tenestene.

1a) (10%)

Teikn klassehierarkiet som består av desse 11 klassene: Object, Kunde, Abonnement, FastTelefon, MobilTelefon, AnalogTelefon, ISDNTelefon, KontantKortTelefon, UngdomsTelefon, PrivatTelefon, PrivatPlussTelefon. Fortel kva for klasse(r) som bør definerast som "abstract".

1b) (15%)

Ei skisse til class Kunde er alt laga, og ser slik ut:

```
class Kunde {    // Eventuelt extends ... , dersom ditt
                // klassehierarki i 1a tilseier det.
                // Eventuelt abstract?
    protected String navn;
    protected String gateadresse;
    protected String postNr;
    protected String postStad;
    protected Vector abonnement;

    public Kunde() { abonnement = new Vector(); }

    public Kunde(String na, String ga, String pn, String ps)
        { ... }

    public void setNavn(String na) { ... }
    public String getNavn() { ... }
    ... Set- og getmetodar for gateadresse, postNr, postStad ...

    public void nyttAbonnement(Abonnement ab) { ... }
    public boolean slettAbonnement(String tlf) { ... }
}
```

Skriv ferdig set- og get-metodar for navn og postNr. Når det gjeld postNr: sørg for at det berre inneheld siffer, og presis 4 siffer. Skriv også ferdig den uferdige konstruktørmetoden. Klassa Abonnement, nevnt i metoden nyttAbonnement, kjem vi attende til seinare.

1c) (10%)

Lag ein equals-metode i class Kunde slik at to kundar er like dersom både navn, gateadresse, postNr og postStad er like.

Lag også metoden nyttAbonnement, den skal sette det nye abonnementet, gitt ved parameteren ab, inn i abonnementvektoren.

Dei andre metodane i klassa kan du anta er ferdige.

Oppgåve 2 (55%)

Alle abonnement har eit telefonnummer, ein vektor som inneheld dei tenester som er utført, men ikkje fakturert, og ein vektor med fakturaer som er sende. Det resulterer i fylgjande skisse:

```
class Abonnement { // Eventuelt extends ...
                  // Eventuelt abstract
    protected String tlfnr;
    protected Vector tenester;
    protected Vector fakturaer;
    ... diverse metodar ...
}
```

Du kan gå ut i frå at det er laga set- og get-metodar for tlfnr.

2a) (10%)

Skriv to konstruktørmetodar for class Abonnement, ein utan parametre, og ein med ein parameter for telefonnummer. Her burde vi testa om telefonnummeret er lovleg, men det kan du droppe. Skriv også equals-metode for class Abonnement, den skal teste likheit på telefonnummeret.

2b) (10%)

Tenestene er registrert som objekt av class Teneste:

```
class Teneste {
    protected int type;           // type teneste
    protected float mengde;      // Kor mykje av den tenesten
    protected Date start;        // Starttidspunkt for bruk av t.
    ... diverse metodar ...
}
```

Du kan anta at set- og get-metodar er laga. Systemet skal kjenne til desse tenestetypene:

Typenummer	Forklaring	Einingsmengde
0	Samtale til fasttelefon	1 minutt varigheit
1	Samtale til mobiltelefon	1 minutt varigheit
2	SMS	1 melding
3	MMS	1 melding
4	WAP	1 MB data

Skriv ein toString-metode for class Teneste. Den skal levere ei line tekst, som inneheld forklaringa i tabellen over, start-tidspunktet, og mengda. Du skal ikkje innføre nye *instansvariable* i class Teneste.

2c) (20%)

Når ein teneste har blitt utført (ein av dei 5 typane) kjenner vi dei tre opplysningane som skal til for å opprette eit Teneste-objekt. Vidare kjenner vi telefonnummeret det er ringt frå. Anta at alle kundar ligg i ein tabell:

```
Kunde[] kundar;
```

Denne tabellen er altså oppretta, og inneheld referanse til kundeobjekt, samla usortert frå starten av tabellen. Antal kundeobjekt er lagra i variabelen "antalKunder". (Dette er i realiteten ein nokså ineffektiv datastruktur.)

Skriv ein metode som registrerer den utførte tenesten:

```
public void registrerTeneste(String tlf, int type,
    float mengde, Date start) { ... }
```

Metoden skal ligge i ei klasse der instansvariablane "kundar" og "antalKunder" er tilgjengelege.

Ved feilsituasjonar kan metoden gjerne kaste unntak. Merk at dette er ein ganske stor jobb, og du kan – og bør – skrive hjelpemetodar i dei klasser du måtte ynskje.

2d) (15%)

Skriv ein metode i class Abonnement:

```
public void fakturer() { ... }
```

Den skal flytte alle tenester i tenester-vektoren over i eit nytt faktura-objekt, og legge fakturaobjektet inn i fakturaer-vektoren. Faktura-klassa ser skissemessig slik ut:

```
class Faktura {
    protected Vector tenester;
    protected Date fakturadato;
    protected float beløp;
    ... evt. fleire metodar ...
}
```

Fakturadatoen skal settast til i dag. Lag den/dei hjelpemetodar du måtte trenge i class Faktura, bortsett frå set- og get-metodar som du kan anta er laga. Tenester-vektoren i abonnementobjektet skal etterpå vere tom. Når du skal finne beløpet på fakturaen kan du ha nytte av å kalle metoden

```
public float pris(Teneste t) { ... }
```

– i class Abonnement. Denne skal du ikkje skrive.

Oppgåve 3 (10%)

Årsaka til at det t.d. er definert fire ulike klasser for ulike typer mobiltelefonabonnement, er at desse kan ha ulike implementasjonar av metoden "pris". Men dersom ein kunde ynskjer å bytte abonnementsordning frå t.d. kontantkort til privat, må ein opprette eit heilt nytt objekt, overføre alle data, og deretter "kaste" det gamle. Ei heilt anna løysing er å nytte "Strategy pattern".

Vis/forklar korleis dette designmønsteret kan nyttast til å gjere klassene AnalogTelefon og ISDNTelefon overflødige: skissér kva for endringar som må gjerast i class FastTelefon, kva for nye klasser du vil trenge, kva desse nye klassene skal innehalde, og korleis dei nyttast. Alt dette skisseaktig, det skal ikkje skrivast komplett kode.

Lykke til!

Eksamen 852 Objektorientert Programmering 07.06.2004 - Bokmål

Råd og retningslinjer. Les oppgaveteksten godt før du går i gang med å løse oppgava. Deloppgavene er uavhengige av hverandre i den forstand at om du ikke får til én oppgave, kan du likevel gjøre neste, som om den første var løst. Fordél tida godt på alle oppgavene. Om du mener en oppgave er upresis, så skriv din egen presisering.

Oppgave 1 (35%)

Du skal utvikle programvare for et telefonselskap. Selskapet har et antall kunder. Hver kunde har null eller flere telefonabonnement. Et telefonabonnement er enten fast-telefon eller mobiltelefon. Fast-telefonen er enten av type "analog" eller "ISDN".

Mobiltelefonabonnementet er enten type "kontantkort", "ungdom", "privat" eller "privat pluss", disse tilbyr litt ulike tjenester og har litt ulike priser på de ulike tjenestene.

1a) (10%)

Tegn klassehierarkiet bestående av disse 11 klassene: Object, Kunde, Abonnement, FastTelefon, MobilTelefon, AnalogTelefon, ISDNTelefon, KontantKortTelefon, UngdomsTelefon, PrivatTelefon, PrivatPlussTelefon. Fortell hvilken eller hvilke klasser som bør defineres som "abstract".

1b) (15%)

En skisse til class Kunde er alt laget, og ser slik ut:

```
class Kunde {    // Eventuelt extends ... , dersom ditt
                // klassehierarki i 1a tilsier det.
                // Eventuelt abstract?
    protected String navn;
    protected String gateadresse;
    protected String postNr;
    protected String postSted;
    protected Vector abonnement;

    public Kunde() { abonnement = new Vector(); }

    public Kunde(String na, String ga, String pn, String ps)
        { ... }

    public void setNavn(String na) { ... }
    public String getNavn() { ... }
    ... Set- og getmetoder for gateadresse, postNr, postSted ...

    public void nyttAbonnement(Abonnement ab) { ... }
    public boolean slettAbonnement(String tlf) { ... }
}
```

Skriv ferdig set- og get-metoder for navn og postNr. Når det gjelder postNr: sørg for at det bare inneholder siffer, og presis 4 siffer. Skriv også ferdig den uferdige konstruktørmetoden. Klassen Abonnement, nevnt i metoden nyttAbonnement, kommer vi tilbake til senere.

1c) (10%)

Lag en equals-metode i class Kunde slik at to kunder er like dersom både navn, gateadresse, postNr og postSted er like.

Lag også metoden nyttAbonnement, den skal sette det nye abonnementet, gitt ved parameteren ab, inn i abonnementvektoren.

De øvrige metodene i klassen kan du anta er ferdige.

Oppgave 2 (55%)

Alle abonnement har et telefonnummer, en vektor som inneholder de tjenester som er utført, men ikke fakturert, og en vektor med sendte fakturaer. Det gir opphav til følgende skisse:

```
class Abonnement { // Eventuelt extends ...
                  // Eventuelt abstract
    protected String tlfnr;
    protected Vector tjenester;
    protected Vector fakturaer;
    ... diverse metoder ...
}
```

Du kan anta at det er laget set- og get-metoder for tlfnr.

2a) (10%)

Skriv to konstruktørmetoder for class Abonnement, en uten parametre, og en med én parameter for telefonnummer. Her burde vi testet om telefonnummeret er lovlig, men det kan du droppe. Skriv også equals-metode for class Abonnement, den skal teste likhet på telefonnummeret.

2b) (10%)

Tjenestene er registrert som objekter av class Tjeneste:

```
class Tjeneste {
    protected int type;           // type tjeneste
    protected float mengde;       // Hvor mye av den tjenesten
    protected Date start;         // Starttidspunkt for bruk av tj.
    ... diverse metoder ...
}
```

Du kan anta at set- og get-metoder er laget. Systemet skal kjenne til disse tjenestetypene:

Typenummer	Forklaring	Enhetsmengde
0	Samtale til fasttelefon	1 minutts varighet
1	Samtale til mobiltelefon	1 minutts varighet
2	SMS	1 melding
3	MMS	1 melding
4	WAP	1 MB data

Skriv en toString-metode for class Tjeneste. Den skal levere en linje tekst, bestående av forklaringen i tabellen over, start-tidspunktet, og mengden. Du skal ikke innføre nye *instansvariable* i class Tjeneste.

2c) (20%)

Når en tjeneste har blitt utført (en av de 5 typene) kjenner vi de tre opplysningene som skal til for å opprette et Tjeneste-objekt. Videre kjenner vi telefonnummeret det er ringt fra. Anta at alle kunder ligger i en tabell:

```
Kunde[] kunder;
```

Denne tabellen er altså opprettet, og inneholder referanse til kundeobjekter, samlet usortert fra starten av tabellen. Antall kundeobjekter er lagret i variabelen "antallKunder". (Dette er i realiteten en nokså ineffektiv datastruktur.)

Skriv en metode som registrerer den utførte tjenesten:

```
public void registrerTjeneste(String tlf, int type,
    float mengde, Date start) { ... }
```

Metoden skal ligge i en klasse der instansvariablene "kunder" og "antallKunder" er tilgjengelige.

Ved feilsituasjoner kan metoden gjerne kaste unntak. Merk at dette er en ganske stor jobb, og du kan – og bør – skrive hjelpemetoder i de klasser du måtte ønske.

2d) (15%)

Skriv en metode i class Abonnement:

```
public void fakturer() { ... }
```

Den skal flytte alle tjenester i tjenester-vektoren over i et nytt faktura-objekt, og legge fakturaobjektet inn i fakturaer-vektoren. Faktura-klassen ser skissemessig slik ut:

```
class Faktura {
    protected Vector tjenester;
    protected Date fakturadato;
    protected float beløp;
    ... evt. flere metoder ...
}
```

Fakturadatoen skal settes til i dag. Lag den/de hjelpemetoder du måtte trenge i class Faktura, bortsett fra set- og get-metoder som du kan anta er laget. Tjenester-vektoren i abonnementobjektet skal etterpå være tom. Når du skal finne beløpet på fakturaen kan du ha nytte av å kalle metoden

```
public float pris(Tjeneste t) { ... }
```

– i class Abonnement. Denne skal du ikke skrive.

Oppgave 3 (10%)

Grunnen til at det f.eks. er definert fire ulike klasser for ulike typer mobiltelefonabonnementer, er at disse kan ha ulike implementasjoner av metoden "pris". Men dersom en kunde ønsker å bytte abonnementsordning fra f.eks. kontantkort til privat, må man opprette et helt nytt objekt, overføre alle data, og deretter "kaste" det gamle. En helt annen løsning er å benytte "Strategy pattern".

Vis/forklar hvordan dette designmønsteret kan brukes til å gjøre klassene AnalogTelefon og ISDNTelefon overflødige: skissér hvilke endringer som må gjøres i class FastTelefon, hvilke nye klasser du vil trenge, hva disse nye klassene skal inneholde, og hvordan de brukes. Alt dette skisseaktig, det skal ikke skrives komplett kode.

Lykke til!