Høgskolen i Telemark

SLUTTPRØVE

5609 OBJEKTORIENTERT PROGRAMMERING

20.12.2005

Tid:	9-12 (9-13 for utvida sluttprøve)
Målform:	Bokmål/nynorsk
Sidetal:	7 (med denne forsida)
Hjelpemiddel:	Alt skriftleg materiale
Merknader:	Ingen
Vedlegg:	Ingen

Eksamensresultata blir offentliggjort på nettet, via Arena høgskole. I tillegg finn du eksamensresultatslister på utsiden av eksamenskontoret, men da trenger du kandidatnummeret ditt, så du bør notere dette på en lapp og legge den i lommeboka.

Avdeling for allmennvitenskaplige fag,

Bokmål

Oppgave 4 er merket "Kun for utvidet sluttprøve".

- De som har bestått begge midtprøvene får 3 timer og skal ikke besvare oppgave 4.
- De som har fritak for midtprøver, kontinuerer, eller ikke har bestått begge midtprøvene får 4 timer og skal besvare alle oppgavene.

Vekting ved sensur:

- Ordinær sluttprøve: Oppgave 1 teller 15%, oppgave 2 teller 25% og oppgave 3 teller 20%. Totalt 60% av samlet karakter i emnet.
- Utvidet sluttprøve: Oppgave 1 teller 20%, oppgave 2 teller 30%, oppgave 3 teller 25% og oppgave 4 teller 25%. Totalt 100% av samlet karakter i emnet.

Oppgave 1-3 (ordinær sluttprøve) tar for seg et felles eksempel hentet fra en bank. Du skal programmere deler av et system for å håndtere bankkonti, kunder og banktransaksjoner (overføring av penger fra en konto til en annen). Oppgavene er uavhengige av hverandre.

Oppgave 1

1-a

En bank ønsker å utvikle en objektorientert modell for representasjon av data om kunder og ulike typer bankkonti. Banken tilbyr både innskuddskonti og lånekonti. En innskuddskonto er enten en brukskonto eller en sparekonto. En lånekonto er enten basert på fast rente eller flytende rente. Sett opp følgende klasser i et hensiktsmessig arvehierarki: *Object*, *Kunde*, *Konto*, *Innskuddskonto*, *Lånekonto*, *Brukskonto*, *Sparekonto*, *FastRenteKonto*, *FlytendeRenteKonto*. Angi hvilke klasser som bør være abstrakte.

1-b

Se på klassen *Konto*. Enhver konto har attributter *kontoNr*, *rentefot*, *navn* (på kunden) og *saldo*. Vis deklarasjon av instansvariable og lag konstruktør(er) for denne klassen. Se deretter på klassen *Sparekonto*. Om en sparekonto skal det i tillegg til fellesattributtene nevnt over også lagres antall uttak så langt i inneværende år. Det er kun tillatt med 4 uttak pr. år. Vis deklarasjon av instansvariable og lag konstruktør(er) for denne klassen.

1-c

Det er ønskelig å kunne ta vare på en samling konto-objekter ved hjelp av ferdiglagede klasser i Java-biblioteket, som for eksempel *ArrayList*, og også kunne sortere slike objektsamlinger. Det er naturlig å bruke kontonummer for å sammenligne to konto-objekter. Forklar kort hva som må til for at konto-objekter kan fungere som elementer i sorterte objektsamlinger. Du skal *ikke* skrive kode her.

Oppgave 2

Banken skal utvikle en web-applikasjon for betaling av regninger ved hjelp av servlett-programmering. Du skal programmere én del av denne applikasjonen.

Du kan i denne oppgaven anta at data om bankkonti og transaksjoner blir lagret i en Access-database med følgende tabeller, der primærnøkler er understreket og fremmednøkler er merket med en stjerne:

- Konto(kontoNr, rentefot, navn, saldo, passord)
- Transaksjon(transId, fraKontoNr*, tilKontoNr, beløp)

Databasen er registrert med logisk ODBC datakildenavn *bank*. Servletten skal kommunisere med databasen ved hjelp av JDBC/ODBC-broen. Kolonnen *transId* er et løpenummer (datatype Autonummer i Access), som innebærer at databasesystemet automatisk genererer verdier ved innsetting av nye rader.

Brukeren må først logge seg inn med brukernavn og passord. Vi tenker oss at hver kunde har kun én konto, og at kontonummeret fungerer som brukernavn. Kontroll med pålogging gjøres av en servlett som du *ikke* skal programmere. Du skal imidlertid forutsette at den oppretter et sesjonsobjekt med navn *kontonr* av typen *String* (hvis pålogging var vellykket).

Etter vellykket pålogging kan brukeren velge å betale regninger. Brukeren får i så fall presentert et HTML-skjema med tekstfelter for å skrive inn kontonummer til mottaker og beløp, samt en knapp for å gjennomføre transaksjonen.

Applikasjonen skal sjekke at beløpet er et positivt tall og at det er dekning på kontoen. I så fall skal det legges til en ny rad i tabellen *Transaksjon*, og en bekreftelse skal bli skrevet ut i en ny nettside. Hvis ikke skal nettsiden vise en passende feilmelding. I praksis vil det være aktuelt å gi brukeren mulighet for å betale flere regninger, gå til hovedmenyen, eller logge ut, men det kan du se bort fra.

Merk at mottaker ikke nødvendigvis er kunde av banken. Det betyr at *tilKontoNr* ikke nødvendigvis er registrert i tabellen *Konto* (og er dermed ikke merket som fremmednøkkel). Applikasjonen skal dermed *ikke* sjekke at dette kontonummeret er lovlig.

Du skal programmere én eller flere servletter som sammen implementerer generering av HTML-skjema og betaling av regning med bekreftelse/feilmelding som beskrevet over.

Oppgave 3

Kun bankens egne ansatte skal kunne opprette og slette bankkonti. Banken ønsker å utvikle en distribuert løsning basert på RMI (Remote Method Invocation) til dette formålet.

Om hver bankkonto kan du i denne oppgaven anta at det blir lagret kontonummer, kundens navn og saldo. For å registrere en ny bankkonto må brukeren skrive inn kontonummer og kundens navn. Saldo blir automatisk satt til 0. For å slette en konto må brukeren skrive inn kontonummeret. Ved forsøk på sletting av en konto som ikke finnes, eller registrering av en

ny konto med et kontonummer som er i bruk skal applikasjonen vise en passende feilmelding.

Applikasjonen skal bestå av en tjenerdel og en klientdel. Du skal *ikke* implementere tjenerdelen. I stedet skal du spesifisere et hensiktsmessig programmeringsgrensesnitt (interface) for tjenerdelen. Deretter kan du anta at tjeneren er ferdig programmert.

Videre skal du lage en skisse av brukergrensesnittet til klientdelen av applikasjonen. Merk figuren med navn og datatype på komponenter (tekstbokser, knapper og lignende). Tegn inn eventuelle paneler og skriv på hvilke layout-håndterere du ville brukt.

Du skal også skrive programkoden for å registrere en ny konto og for å slette en eksisterende konto. Gjør rede for egne forutsetninger hvis oppgaveteksten på noe punkt er uklar eller ufullstendig.

Oppgave 4 (Kun for utvidet sluttprøve)

Du skal programmere en såkalt *prioritetskø* ved hjelp av en enkeltkjedet liste. I en prioritetskø blir elementene tilordnet en prioritet (et positivt heltall) ved innsetting, og det er alltid den med høyest prioritet som blir tatt ut først. Hvis flere elementer har samme prioritet skal den som har ventet lengst bli tatt ut først.

Køen skal bygges opp fra klassene *Node* og *PrioKø*. Klassen *Node* skal inneholde et heltall prioritet, en referanse til neste *Node* og en referanse til selve elementet. Klassen *PrioKø* skal inneholde en referanse til den første noden.

Betrakt følgende utdrag fra et testprogram, der *PK* er en objektvariabel av klassen *PrioKø*:

```
PK.settInn( "per" , 3 );              // per settes inn med prioritet 3
PK.settInn( "pål" , 2 );
PK.settInn( "espen" , 2 );
PK.settInn( "grete" , 3 );
System.out.println( PK.taUt() );
System.out.println( PK.taUt() );
```

Koden vil skrive ut "per" først og så "grete". Rett etter innsettingsoperasjonene (før utskrift) vil elementene være ordnet som følger: per, grete, pål, espen.

Programmer klassene *Node* og *PrioKø*. Det er ikke nødvendig å programmere set- og get-metoder i klassen *Node*. Som et minimum skal klassene håndtere elementer av type *String*. Prøv imidlertid å lage løsningen så generell som mulig.

--- Slutt på oppgavesett ---

Nynorsk

Oppgåve 4 er merka "Berre for utvida sluttprøve".

- Dei som har stått på begge midtprøvene får 3 timar og skal ikkje svare på oppgåve 4.
- Dei som har fritak for midtprøver, kontinuerer, eller ikkje har stått på begge midtprøvene får 4 timar og skal svare på alle oppgåvene.

Vekting ved sensur:

- Ordinær sluttprøve: Oppgåve 1 tel 15%, oppgåve 2 tel 25% og oppgåve 3 tel 20%. Totalt 60% av samla karakter i emnet.
- Utvida sluttprøve: Oppgåve 1 tel 20%, oppgåve 2 tel 30%, oppgåve 3 tel 25% og oppgåve 4 tel 25%. Totalt 100% av samla karakter i emnet.

Oppgåve 1-3 (ordinær sluttprøve) tek for seg eit felles døme henta frå ein bank. Du skal programmere delar av eit system for å handtere bankkonti, kundar og banktransaksjonar (overføring av pengar frå ein konto til ein annan). Kvar oppgåve er sjølvstendig.

Oppgåve 1

1-a

Ein bank ønskjer å utvikle ein objektorientert modell for representasjon av data om kundar og ulike slag bankkonti. Banken tilbyr både innskuddskonti og lånekonti. Ein innskuddskonto er anten ein brukskonto eller ein sparekonto. Ein lånekonto er anten basert på fast rente eller flytande rente. Set opp følgjande klasser i eit formålstenleg arvehierarki: *Object*, *Kunde*, *Konto*, *Innskuddskonto*, *Lånekonto*, *Brukskonto*, *Sparekonto*, *FastRenteKonto*, *FlytendeRenteKonto*. Peik på kva for klasser som bør vere abstrakte.

1-b

Sjå på klassa *Konto*. Einkvar konto har attributt *kontoNr*, *rentefot*, *namn* (på kunden) og *saldo*. Skriv deklarasjon av instansvariablar og konstruktør(ar) for denne klassa. Sjå deretter på klassa *Sparekonto*. Om ein sparekonto skal det i tillegg til fellesattributta nemnt over óg lagrast talet på uttak så langt i inneverande år. Det er berre tillete med 4 uttak pr. år. Skriv deklarasjon av instansvariablar og konstruktør(ar) for denne klassa.

1-c

Det er ønskjeleg å kunne ta vare på ein samling konto-objekt ved hjelp av ferdiglaga klasser i Java-biblioteket, som til dømes *ArrayList*, og dessutan å kunne sortere slike objektsamlingar. Det er naturleg å nytte kontonummer for å samanlikne to konto-objekt. Forklar kort kva som må til for at konto-objekt kan brukast som element i sorterte objektsamlingar. Du skal *ikkje* skrive kode her.

Oppgåve 2

Banken skal utvikle ein web-applikasjon for betaling av rekningar ved hjelp av servlett-programmering. Du skal programmere ein del av denne applikasjonen.

Du kan i denne oppgåva gå ut ifrå at data om bankkonti og transaksjonar vert lagra i ein Access-database med følgjande tabellar, der primærnøklar er understreka og framandnøklar er merkte med ei stjerne:

- Konto(kontoNr, rentefot, namn, saldo, passord)
- Transaksjon(transId, fråKontoNr*, tilKontoNr, beløp)

Databasen er registrert med logisk ODBC datakildenamn *bank*. Servletten skal kommunisere med databasen ved hjelp av JDBC/ODBC-brua. Kolonna *transId* er eit løpenummer (datatype Autonummer i Access), som tyder at databasesystemet automatisk genererer verdiar ved innsetjing av nye rader.

Brukaren må først logge seg inn med brukarnamn og passord. Vi tenker oss at kvar kunde har berre ein konto, og at kontonummeret vert nytta som brukarnamn. Kontroll med pålogging gjerast av ein servlett som du *ikkje* skal programmere. Du skal derimot føresetje at den opprettar eit sesjonsobjekt med namn *kontonr* av typen *String* (viss pålogging var vellukka).

Etter vellukka pålogging kan brukaren vele å betale rekningar. Brukaren får i så fall presentert eit HTML-skjema med tekstfelt for å skrive inn kontonummer til mottakar og beløp, samt ein knapp for å gjennomføre transaksjonen.

Applikasjonen skal sjekke at beløpet er eit positivt tal og at det er dekning på kontoen. I så fall skal det leggjast til ein ny rad i tabellen *Transaksjon*, og ein kvittering skal bli skreve ut i ein ny nettside. Viss ikkje skal nettsida syne ein passande feilmelding. I praksis vil det vere aktuelt å gi brukaren sjansen til å betale fleire rekningar, gå til hovudmenyen, eller logge ut, men det kan du sjå bort ifrå.

Merk at mottakar ikkje alltid er kunde av banken. Det tyder at *tilKontoNr* ikkje alltid er registrert i tabellen *Konto* (og er dermed ikkje merka som framandnøkkel). Applikasjonen skal dermed *ikkje* sjekke at dette kontonummeret er lovleg.

Du skal programmere ein eller fleire servlettar som saman implementerer generering av HTML-skjema og betaling av rekning med kvittering/feilmelding som forklart over.

Oppgåve 3

Berre dei tilsette i banken skal kunne opprette og slette bankkonti. Banken ønskjer å utvikle ei distribuert løysing bygd på RMI (Remote Method Invocation) til dette føremålet.

Om kvar bankkonto kan du i denne oppgåva gå ut ifrå at det blir lagra kontonummer, kundens namn og saldo. For å registrere ein ny bankkonto må brukaren skrive inn kontonummer og kundens namn. Saldo blir automatisk satt til 0. For å slette ein konto må brukaren skrive inn kontonummeret. Ved forsøk på sletting av ein konto som ikkje finst, eller registrering av ein

ny konto med eit kontonummer som er i bruk skal applikasjonen syne ein passande feilmelding.

Applikasjonen skal byggast opp ifrå ein tenardel og ein klientdel. Du skal *ikkje* implementere tenardelen. I staden skal du spesifisere eit føremålstenleg programmeringsgrensesnitt (interface) for tenardelen. Etterpå kan du gå ut ifrå at tenaren er ferdig programmert.

Vidare skal du lage ei skisse av brukargrensesnittet til klientdelen av applikasjonen. Merk figuren med namn og datatype til komponentar (tekstboksar, knappar og liknande). Teikn inn eventuelle panelar og skriv på kva for layout-handterar du ville nytta.

Du skal óg skrive programkoden for å registrere ein ny konto og for å slette ein eksisterande konto. Gjer greie for eigne føresetnader viss oppgåveteksten på noko punkt er uklar eller ufullstendig.

Oppgåve 4 (Berre for utvida sluttprøve)

Du skal programmere ein såkalla *prioritetskø* ved hjelp av ei enkeltkjeda liste. I ein prioritetskø blir elementa tilordna ein prioritet (eit positivt heiltal) ved innsetjing, og det er alltid den med høgast prioritet som blir tatt ut først. Viss fleire element har same prioritet skal den som har venta lengst bli tatt ut først.

Køen skal byggjast opp frå klassane *Node* og *PrioKø*. Klassa *Node* skal innehelde eit heiltal prioritet, ein referanse til neste *Node* og ein referanse til sjølve elementet. Klassa *PrioKø* skal innehelde ein referanse til den første noden.

Sjå på følgjande utdrag frå eit testprogram, der *PK* er ein objektvariabel av klassa *PrioKø*:

```
PK.settInn( "per" , 3 );          // per vert satt inn med prioritet 3
PK.settInn( "pål" , 2 );
PK.settInn( "espen" , 2 );
PK.settInn( "grete" , 3 );
System.out.println( PK.taUt() );
System.out.println( PK.taUt() );
```

Koden vil skrive ut "per" først og så "grete". Rett etter innsettingsoperasjonane (før utskrift) vil elementa vere ordna som følgjer: per, grete, pål, espen.

Programmer klassene *Node* og *PrioKø*. Det er ikkje naudsynt å programmere set- og get-metodar i klassa *Node*. Som eit minimum skal klassene handtere element av type *String*. Freist derimot å lage løysinga så generell som mogleg.

--- Slutt på oppgåvesett ---