Høgskolen i Telemark

EKSAMEN

5609 OBJEKTORIENTERT PROGRAMMERING

08.12.2008

Tid:	4 timer, 9.00 – 13.00
Målform:	Bokmål / nynorsk (programkode er ikke oversatt)
Sidetall:	Forside + 5 + 5
Hjelpemidler:	Alle trykte og skrevne (IKKE elektroniske) Besvarelsen kan skrives med tekstbehandlingsprogram.
Merknader:	Ingen
Vedlegg:	Ingen

Eksamensresultatene blir offentliggjort på nettet, via Arena høgskole. I tillegg finner du eksamensresultatslister på utsiden av eksamenskontoret, men da trenger du kandidatnummeret ditt. Du bør notere dette på en lapp som du tar vare på.

Besvarelsen kan skrives med tekstbehandlingsprogram (Word eller Notisblokk), uten syntakshjelp for Java. Hele besvarelsen skal skrives som ett dokument (fil) for å lette utskrift.

Besvarelsen skal leveres på papir, dvs. som utskrift fra skriver. Eksamensvaktene vil hjelpe til med utskrift. Du må selv se gjennom og kontrollere utskriften. Du er ansvarlig for at det som leveres inn er komplett. Figurer og tegningen som du har tegnet for hånd legges ved besvarelsen ved innlevering. Tid til utskrift og kontroll regnes ikke inn i eksamenstiden.

Bokmål

Oppgavene er uavhengige: Selv om det er én oppgave du ikke har løst, kan du løse neste som om den forrige var løst. Disponer tiden godt, slik at du får gjort noe på alle oppgavene. Om en oppgave er uklar så lag dine egne forutsetninger, og forklar disse.

Du skal lage noen Java-klasser som kan brukes i programmer for kortspill. I oppgave 2 og 3 skal du også programmere et enkelt kortspill **NB! Du trenger ingen forkunnskap om kortspillet for å løse denne oppgaven.**

Faktarute: kortstokk. En kortstokk inneholder 52 vanlige kort, og eventuelt jokere. I denne oppgaven ser vi bort fra joker. Hvert kort har en "*farge*" som kan være **spar**, **hjerter**, **ruter** eller **kløver**. Farge har altså ikke noe med rødt eller svart å gjøre. Hvert kort har også en "*verdi*" som er en av disse 13: **Ess, 2, 3, 4, 5, 6, 7, 8, 9, 10, knekt, dame, konge**. I denne oppgaven har esset alltid verdien 1. Sekvensen 2, 3, ..., konge er i stigende rekkefølge. Totalt har en kortstokk altså 4 * 13 = 52 ulike kort.

Oppgave 1 (50 %)

Programmet skal inneholde en klasse **Kort**. Et objekt av denne klassen representerer ett kort i en kortstokk. Etter grundig analyse, har du kommet fram til følgende skisse av class Kort (linjene er nummerert):

```
1.   public class Kort implements Comparable<Kort> {
2.     public final static int KLØVER = 0;
3.     public final static int RUTER = 1;
4.     public final static int HJERTER = 2;
5.     public final static int SPAR = 3;
6.
7.     public final static int ESS = 1;
8.     public final static int KNEKT = 11;
9.     public final static int DAME = 12;
10.    public final static int KONGE = 13;
11.
12.    protected int farge;
13.    protected int verdi;
14.
15.    public Kort(int farge, int verdi) { ... }
16.    public int getFarge() { ... }
17.    public int getVerdi() { ... }
18.    public boolean erBildeKort() { ... }
19.    public String toString() { ... }
20.    public boolean equals(Kort k) { ... }
21.    public int compareTo(Kort k) { ... }
22.  }
```

Oppgave 1a

Programmer ferdig klassen Kort i samsvar med skissen over, og punktene nedenfor. NB! Du behøver ikke skrive av linje 1-13 i koden over!

- Du skal ikke introdusere nye instansvariable.
- Ved ulovlige parameterverdier til konstruktøren, skal den kaste et unntaks av klassen `IllegalArgumentException`.
- Metoden `erBildeKort` skal returnere `true` hvis kortet er ess, knekt, dame eller konge, ellers `false`.
- Metoden `toString` skal returnere f.eks. "Ruter 9", "Kløver ess" osv.
- Metoden `equals()` skal returnere true hvis to kort er like, dvs. lik verdi og lik farge, ellers false.
- Metoden `compareTo()` "arves" fra grensesnittet `java.lang.Comparable`. Du må lage en implementasjon av metoden i klassen Kort. I denne oppgaven er et kort "mindre" enn et annet kort hvis verdien er mindre, og større hvis verdien er større. Vi bryr oss altså ikke om fargen på kortet når vi sammenlikner kort med denne metoden.

Oppgave 1b

Forklar kort hvorfor vi ikke skal ha metodene setFarge og setVerdi i klassen Kort.

Oppgave 1c

I programmet har vi behov for to typer samling av kort, nemlig 1) kortstokken (før, under eller etter utdeling til spillerne), og 2) de "hendene" spillerne har, dvs. de kortene hver spiller sitter med i handa. Disse to kortsamlingene ser ut til å bli ganske like. Du skal derfor lage en generell abstrakt klasse for kortsamlinger. Skjellettet til denne klassen ser slik ut:

```
public abstract class Kortsamling {
  protected ArrayList<Kort> kortene;

  public Kortsamling() { ... }
  public boolean inneholder(int farge, int verdi) { ... }
  public Kort taUt(int farge, int verdi) { ... }
  public void settInn(Kort k) { ... }
  public int antall(){ ... }
}
```

Programmer ferdig klassen Kortsamling i samsvar med skissen over, og punktene nedenfor:

- Konstruktøren skal opprette en ny tom kortsamling, dvs. uten noen kort.
- Metoden `inneholder`, skal returnere `true` dersom samlingen inneholder kortet med den oppgitte farge og verdi, ellers `false`.
- Metoden `taUt` skal ta kortet med oppgitt farge og verdi ut av samlingen, og returnere kortet, dersom det finnes i samlingen. Hvis kortet ikke finnes i samlingen skal metoden returnere null.
- Metoden `settInn` skal sette det aktuelle kortet inn i samlingen. Vi bryr oss ikke noe om hvor i samlingen kortet blir plassert. Hvis kortet allerede finnes i samlingen, skal metoden i stedet kaste et passende unntak.
- Metoden `antall` skal returnere antall kort i samlingen.

Oppgave 1d

Lag en ny klasse Kortstokk som er en subklasse av Kortsamling, og som oppfyller disse punktene:

- Konstruktøren for klassen Kortstokk skal lage de 52 kortene og legge disse inn i kortstokken. Bruk løkker, ikke 52 enkeltsetninger!
- Klassen skal ha en metode `public Kort taUtTilfeldig()`. Metoden skal ta ut ett tilfeldig kort av kortstokken, og returnere dette. Metoden må oppføre seg fornuftig også dersom det ikke er flere kort igjen i samlingen. Tips: Metoden `Math.random()` returnerer en tilfeldig double verdi mellom 0.0 og 1.0.

Oppgave 1e

Du skal programmere en ny klasse Hand som også er en subklasse av Kortsamling. Klassen representerer en spillers hand, dvs. de kortene spilleren har til en hver tid.

- Når et nytt objekt av klassen Hand opprettes, skal handa ikke inneholde noen kort.
- Klassen skal ha en metode `bildeKort` som returnerer antall bildekort på handa.
- Klassen skal ha en metode som sorterer kortene på hånden etter verdi.

Oppgave 1f

Utvid klassen Kortstokk med en ny metode:

```
public Hand delUt(int antKort) { ... }
```

Metoden skal returnere et objekt av klassen Hand. Objektet skal inneholde så mange kort som er angitt i parameteren til metoden. Kortene skal trekkes tilfeldig fra kortstokken (og selvfølgelig fjernes fra kortstokken). Metoden skal kaste et unntak dersom det ikke er nok kort igjen i kortstokken.

Oppgave 1g

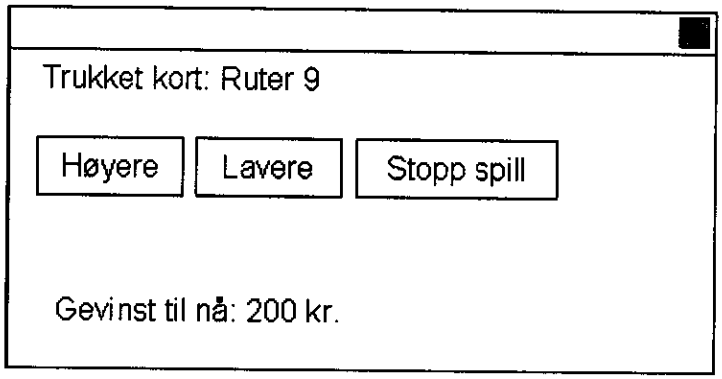
Tegn et UML klassediagram som beskriver hele oppgave 1. Du behøver ikke ta med statiske variable/konstanter i diagrammet.

Oppgave 2 (30%)

Du skal nå lage et enkelt kortspill for *en* spiller.

- Spillet starter med at det trekkes et tilfeldig kort som vises for spilleren.
- Så skal spilleren gjette på om neste kort blir høyere eller lavere *i verdi* enn dette. Fargen skal man ikke ta hensyn til.
- Deretter trekkes et nytt tilfeldig kort og dette sammenlignes med det forrige. (Ess skal tolkes som det laveste kortet).
    - Dersom det nye kortet har samme verdi som det gamle, skal det alltid godkjennes.
    - Dersom spilleren gjettet feil, går hun ut av spillet uten gevinst.
    - Dersom hun gjettet rett, kan hun velge å stoppe spillet med oppnådd gevinst, eller gå videre til ny runde, ved å fortsette å gjette på høyere/lavere.
- For hver runde dobles gevinsten, som starter på 100 kroner.
- Etter maksimalt 10 runder kan man vinne 51.200 kroner. Spillet skal da uansett stoppes av programmet.
- Programmet skal hele tiden vise hvor stor gevinsten er for brukeren.

Programmet skal lages som en Java applikasjon med et Swing-basert GUI slik skissen nedenfor viser:



Programmer GUI'et og logikken i programmet slik det er beskrevet over.

Tips: Du behøver ikke klassen Hand i dette programmet.

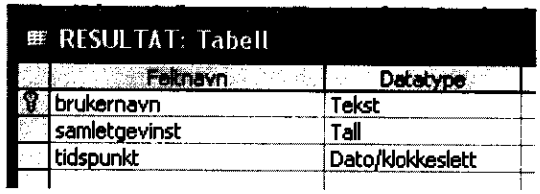
Oppgave 3 (20%)

Vi fortsetter på spillprogrammet fra oppgave 2.

Når et spill stoppes (av spilleren eller programmet), skal programmet ta vare på spillerens samlede gevinst i en database. Hver gang brukeren vinner / stopper et spill **med gevinst**, skal programmet spørre brukeren om et brukernavn (skal ikke programmeres). Programmet skal deretter lagre oppnådd gevinst knyttet til dette navnet i databasen.

Dataene skal lagres i en databasetabell i Access med følgende tabelldefinisjon:

```
create table RESULTAT (
  brukernavn     Text PRIMARY KEY,
  samletgevinst  Number,
  tidspunkt      DateTime
);
```



- Kolonnen `brukernavn` er primærnøkkel i tabellen RESULTAT. Tabellen inneholder altså maksimalt én rad for hvert brukernavn.
- Kolonnen `samletgevinst` skal inneholde brukerens samlede gevinst så langt. Hvis brukeren spiller (og vinner) flere ganger, skal gevinstene altså summeres opp i denne kolonnen.
- Kolonnen `tidspunkt` skal inneholde dato og klokkeslett når gevinsten sist ble lagret/endret.

Det er laget en ODBC-datakilde til databasen, med datakildenavn *KORTSPILL* (DSN-navn). Databasen krever ikke passord/brukernavn for pålogging.

Lagring i databasen skal utføres av klassemetoden lagreResultat() i klassen dbKontroller nedenfor. Programmer ferdig metoden. NB! Du behøver ikke programmere endringene i GUIet som skal til for å lagre gevinsten.

```
public class dbKontroller {
    static final String odbcDriver=
                        "sun.jdbc.odbc.JdbcOdbcDriver";
    static final String dbURL="jdbc:odbc:KORTSPILL";

    public static boolean lagreResultat(
              String brukernavn, int nyGevinst)
     {.....}

}
```

Slutt på oppgavesettet (bokmål)

Nynorsk

Oppgåvene er uavhengige: Sjølv om det er ei oppgåve du ikkje har løyst, kan du løyse neste som om den førre var løyst. Disponer tida godt, slik at du får gjort noko på alle oppgåvene. Om ei oppgåve er uklår så lag dine eigne føresetnader, og forklar desse.

Du skal lage nokre Java-klasser som kan nyttast i programmer for kortspel. I oppgåve 2 og 3 skal du òg programmere eit enkelt kortspel **NB! Du treng ingen forkunnskapar om kortspelet for å løyse denne oppgåva.**

Faktarute: kortleik. Ein kortleik inneheld 52 vanlege kort, og eventuelt joker. I denne oppgåva ser vi bort frå joker. Kvart kort har ein "*farge*" som kan være **spar**, **hjerter**, **ruter** eller **kløver**. Farge har altså ikkje noko med raudt eller svart å gjere. Kvart kort har òg ein "*verdi*" som er ein av desse 13: **Ess, 2, 3, 4, 5, 6, 7, 8, 9, 10, knekt, dame, konge**. I denne oppgåva har esset alltid verdien 1. Sekvensen 2, 3, ..., konge er i stigande rekkefylje. Totalt har ein kortleik altså 4 * 13 = 52 ulike kort.

Oppgåve 1 (50 %)

Programmet skal innehalde ei klasse **Kort**. Eit objekt av denne klassa representerer eit kort i ein kortleik. Etter grundig analyse, har du kome fram til fyljande skisse av class Kort (linene er nummererte):

```
23.  public class Kort implements Comparable<Kort> {
24.    public final static int KLØVER = 0;
25.    public final static int RUTER = 1;
26.    public final static int HJERTER = 2;
27.    public final static int SPAR = 3;
28.
29.    public final static int ESS = 1;
30.    public final static int KNEKT = 11;
31.    public final static int DAME = 12;
32.    public final static int KONGE = 13;
33.
34.    protected int farge;
35.    protected int verdi;
36.
37.    public Kort(int farge, int verdi) { ... }
38.    public int getFarge() { ... }
39.    public int getVerdi() { ... }
40.    public boolean erBildeKort() { ... }
41.    public String toString() { ... }
42.    public boolean equals(Kort k) { ... }
43.    public int compareTo(Kort k) { ... }
44.  }
```

Oppgåve 1a

Programmer ferdig klassa Kort i samsvar med skissa over, og punkta nedanfor. NB! Du treng ikkje skrive av line 1-13 i koden over!

- Du skal ikkje introdusere nye instansvariable.
- Ved ulovlege parameterverdiar til konstruktøren, skal den kaste eit unntak av klassa `IllegalArgumentException`.
- Metoden `erBildeKort` skal returnere `true` dersom kortet er ess, knekt, dame eller konge, elles `false`.
- Metoden `toString` skal returnere f.eks. "Ruter 9", "Kløver ess" osv.
- Metoden `equals()` skal returnere true dersom to kort er like, dvs. lik verdi og lik farge, elles false.
- Metoden `compareTo()` "arves" frå grensesnittet `java.lang.Comparable`. Du må lage ein implementasjon av metoden i klassa Kort. I denne oppgåva er eit kort "mindre" enn eit anna kort dersom verdien er mindre, og større dersom verdien er større. Vi bryr oss altså ikkje om fargen på kortet når vi samanlikner kort med denne metoden.

Oppgåve 1b

Forklar kort kvifor vi ikkje skal ha metodane setFarge og setVerdi i klassa Kort.

Oppgåve 1c

I programmet har vi behov for to typar samling av kort, nemlig 1) kortleiken (før, under eller etter utdeling til spelarane), og 2) dei "hendene" spelarane har, dvs. dei korta kvar speler sitter med i handa. Desse to kortsamlingane ser ut til å bli ganske like. Du skal derfor lage ei generell abstrakt klasse for kortsamlingar. Skjelettet til denne klassa ser slik ut:

```
public abstract class Kortsamling {
  protected ArrayList<Kort> korta;

  public Kortsamling() { ... }
  public boolean inneheld(int farge, int verdi) { ... }
  public Kort taUt(int farge, int verdi) { ... }
  public void settInn(Kort k) { ... }
  public int antall(){ ... }
}
```

Programmer ferdig klassa Kortsamling i samsvar med skissa over, og punkta nedanfor:

- Konstruktøren skal opprette ei ny tom kortsamling, dvs. utan nokon kort.
- Metoden `inneheld` skal returnere `true` dersom samlinga inneheld kortet med den oppgitte farge og verdi, elles `false`.
- Metoden `taUt` skal ta kortet med oppgitt farge og verdi ut av samlinga, og returnere kortet, dersom det finnes i samlinga. Dersom kortet ikkje finnes i samlinga skal metoden returnere null.
- Metoden `settInn` skal sette det aktuelle kortet inn i samlinga. Vi bryr oss ikkje noko om kor i samlinga kortet blir plassert. Dersom kortet allereie finnes i samlinga, skal metoden i staden kaste eit passende unntak.
- Metoden `antall` skal returnere tal på kort i samlinga.

Oppgåve 1d

Lag ei ny klasse Kortleik som er ei subklasse av Kortsamling, og som oppfyller desse punkta:

- Konstruktøren for klassa Kortleik skal lage dei 52 korta og legge desse inn i kortleiken. Bruk løkker, ikkje 52 enkeltsetningar!
- Klassa skal ha ein metode `public Kort taUtTilfeldig()`. Metoden skal ta ut eit tilfeldig kort av kortleiken, og returnere dette. Metoden må oppføre seg fornuftig òg dersom det ikkje er fleire kort igjen i samlinga. Tips: Metoden `Math.random()` returnerer ein tilfeldig double verdi mellom 0.0 og 1.0.

Oppgåve 1e

Du skal programmere ei ny klasse Hand som òg er ei subklasse av Kortsamling.
Klassa representerer ein spelars hand, dvs, dei korta spelaren har til ein kvar tid.

- Når eit nytt objekt av klassa Hand vert oppretta, skal handa ikkje innehalde nokon kort.
- Klassa skal ha ein metode `bildeKort` som returnerer tal på bildekort på handa.
- Klassa skal ha ein metode som sorterer korta på handa etter verdi.

Oppgåve 1f

Utvid klassa Kortleik med ein ny metode:

```
public Hand delUt(int antKort) { ... }
```

Metoden skal returnere eit objekt av klassa Hand. Objektet skal innehalde så mange kort som er gjeve i parameteren til metoden. Korta skal trekkast tilfeldig frå kortleiken (og sjølvsagt fjernast frå kortleiken). Metoden skal kaste eit unntak dersom det ikkje er nok kort igjen i kortleiken.

Oppgåve 1g

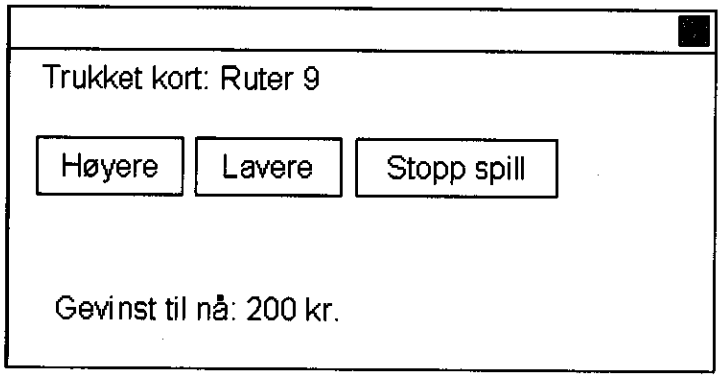
Teikn eit UML klassediagram som beskriver heile oppgåve 1. Du treng ikkje ta med statiske variablar/konstantar i diagrammet.

Oppgåve 2 (30%)

Du skal no lage eit enkelt kortspel for *ein* spelar.

- Spelet startar med at det vert trekt eit tilfeldig kort som vert vist for spelaren.
- Så skal spelaren gjette på om neste kort blir høgare eller lågare *i verdi* enn dette. Fargen skal ein ikkje ta omsyn til.
- Deretter vert det trekt eit nytt tilfeldig kort og dette vert samanlikna med det førre. (Ess skal tolkast som det lågaste kortet).
    - Dersom det nye kortet har same verdi som det gamle, skal det alltid godkjennast.
    - Dersom spelaren gissa feil, går hun ut av spelet utan vinst.
    - Dersom hun gissa rett, kan hun velje å stoppe spelet med oppnådd vinst, eller gå vidare til ny runde, ved å fortsette å gjette på høgare/lågare.
- For kvar runde vert vinsten dobla. Vinsten startar på 100 kroner.
- Etter maksimalt 10 rundar kan ein vinne 51.200 kroner. Spelet skal da uansett stoppast av programmet.
- Programmet skal heile tida vise kor stor vinsten er for brukaren.

Programmet skal lages som ein Java applikasjon med eit Swing-basera GUI slik skissa nedanfor syner:



Programmer GUI'et og logikken i programmet slik det er omtala over.

Tips: Du treng ikkje klassa Hand i dette programmet.

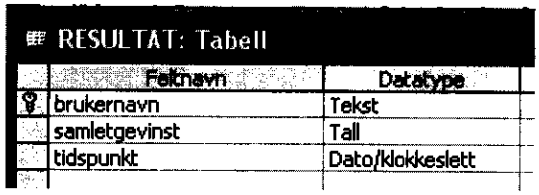
Oppgåve 3 (20%)

Vi fortsetter på spelprogrammet frå oppgåve 2.

Når eit spel vert stoppa (av spelaren eller programmet), skal programmet ta vare på spelarens samla vinst i ein database. Kvar gong brukaren vinner / stoppar eit spel **med vinst**, skal programmet spørje brukaren om eit brukarnamn (skal ikkje programmerast). Programmet skal deretter lagre oppnådd vinst knytt til dette namnet i databasen.

Data skal lagrast i ein databasetabell i Access med fyljande tabelldefinisjon:

```
create table RESULTAT (
  brukernavn     Text PRIMARY KEY,
  samletgevinst  Number,
  tidspunkt      DateTime
);
```



- Kolonnen `brukernavn` er primærnøkkel i tabellen RESULTAT. Tabellen inneheld altså maksimalt ei rad for kvart brukarnamn.
- Kolonnen `samletgevinst` skal innehalde brukarens samla vinst så langt. Dersom brukaren speler (og vinner) fleire gonger, skal vinstane altså summerast opp i denne kolonnen
- Kolonnen `tidspunkt` skal innehalde dato og klokkeslett når vinsten sist vart lagra/endra.

Det er laget ei ODBC-datakjelde til databasen, med datakjeldenamn *KORTSPEL* (DSN-namn). Databasen krev ikkje passord/brukarnamn for pålogging.

Lagring i databasen skal utførast av klassemetoden lagreResultat() i klassa dbKontroller nedanfor. Programmer ferdig metoden. NB! Du treng ikkje programmere endringane i GUI'et som skal til for å lagre vinsten.

```
public class dbKontroller {
    static final String odbcDriver=
                        "sun.jdbc.odbc.JdbcOdbcDriver";
    static final String dbURL="jdbc:odbc:KORTSPEL";

    public static boolean lagreResultat(
              String brukernamn, int nyVinst)
     {.....}

}
```

Slutt på oppgåvene (nynorsk)