



Høgskolen i Telemark

Avdeling for allmenne fag

EKSAMEN

I

852 OBJEKTORIENTERT PROGRAMMERING

04.12.2001

Tid: 5 timar (0900 – 1400)

Målform: Nynorsk/bokmål

Sidetal: 4 + 4

Hjelpemiddel: Alle trykte og skrevne

Merknader: Ingen

Vedlegg: Ingen

Eksamen 852 Objektorientert Programmering 04.12.2001 - Nynorsk

Oppgåvene er uavhengige av kvarandre i den meining at sjøl om du ikkje har løyst éi oppgåve, kan du løyse neste som om forrige var løyst. Disponér tida godt, slik at du får gjort noko på alle oppgåvene. Om ei oppgåve er uklår så lag dine egne vilkår, og forklar desse.

Oppgåve 1 (60%)

Ein venn på øk/adm studia har fått ein genial forretningsidé: å la publikum spele kort på internett. Du har fått oppdraget med å utvikle dette. Det skal lagast fleire slag kortspel, så det fyrste som skal lagast er eit generelt rammeverk for kortspel.

Faktarute : kortstokk. Ein kortstokk inneheld 52 vanlege kort, og eventuelt jokerar. I denne oppgåva ser vi bort frå joker. Kwart kort har ein "farge" som kan vere spar, hjerter, ruter eller kløver, og ein "verdi" som er ein av: Ess, 2, 3, 4, 5, 6, 7, 8, 9, 10, knekt, dame, konge. Sekvensen 2, 3, ..., konge er i stigande rekkefylgje, men esset ser ein stundom som det minste kortet (som 1), og stundom som det største (som 14). Totalt har vi $4 * 13 = 52$ ulike kort.

Du slår fast at dette rammeverket må innehalde class Kort, eit objekt av denne klassa representerer eitt kort i ein kortstokk. Vidare ser du at det er to ulike typer av samling av kort, nemleg 1) kortstokken før, under eller etter utdeling til spelarane, og 2) dei "hendene" spelarane har, dei korta kvar spelar sit med i handa. Desse to kortsamlingane ser ut til å bli ganske like. Du bestemmer deg for fylgjande klassehierarki:

```
public class Kortsamling { ... }
public class Kortstokk extends Kortsamling { ... }
public class Hand extends Kortsamling { ... }
```

Etter grundig analyse, har du kome fram til fylgjande skisse av class Kort og class Kortsamling:

```
public class Kort {
    public final static int KLØVER = 0;
    public final static int RUTER = 1;
    public final static int HJERTER = 2;
    public final static int SPAR = 3;
    public final static int ESS = 1;
    public final static int KNEKT = 11;
    public final static int DAME = 12;
    public final static int KONGE = 13;

    protected int farge;
    protected int verdi;

    public Kort(int farge, int verdi) { ... }
    public int getFarge() { ... }
    public int getVerdi() { ... }
    public String toString() { ... }
}
```

```

public class Kortsamling {
    protected Vector korta;

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

```

Oppgåve 1a

Lag ferdig class Kort i samsvar med skissa over. Du skal ikkje introdusere nye instansvariable, det gjeld også i dei andre klassene. Ved ulovlege parameterverdiar i denne og andre klasser - kaste gjerne ein `IllegalArgumentException`. `toString`-metoden skal returnere t.d. "Ruter 9", "Kløver ess". Forklar kort kvifor vi ikkje skal ha metodane `setFarge` og `setVerdi`.

Oppgåve 1b

Lag ferdig class Kortsamling i samsvar med skissa over. Metoden `inneheld` skal returnere `true` dersom samlinga inneheld kortet med den farge og den verdi, `false` ellers. Metoden `taUt` skal ta kortet ut av vektoren og returnere kortet dersom det finst i vektoren, kaste unntak dersom det ikkje finst. Metoden `settInn` skal sette dette kortet inn i samlinga, bakerst i vektoren. `tom` skal returnere `true` dersom samlinga er tom, `false` ellers. `antal` skal returnere antal kort i samlinga.

Oppgåve 1c

Fylgjande skisser er laga til class `Kortstokk` og class `Hand`:

```

public class Kortstokk extends Kortsamling {
    protected boolean stokka;

    public Kortstokk() { ... }
    public void stokke() { ... }
    public Hand delUt(int antKort) { ... }
    public Kort taUtTilfeldig() { ... }
    public Kort taUtFyrste() { ... }
    public boolean isStokka() { ... }
}

public class Hand extends Kortsamling {
    public Hand() { ... }
    public void sorter() { ... }
    public int билетKort() { ... }
}

```

Kortstokk: Instansvariablen `stokka` skal halde styr på om korta i kortstokken er stokka, dvs. om dei ligg i tilfeldig rekkefylgje. Konstruktøren skal initialisere instansvariablane, herunder lage dei 52 korta (ved hjelp av løkker, ikkje 52 setningar). `stokke` skal stokke om dei korta som finst slik at dei ligg i tilfeldig rekkefylgje. Opprett gjerne ein ekstra vektor og

flytt korta over dit i tilfeldig rekkefyllgje ved hjelp av ein random-funksjon. `delUt` skal lage og returnere ei hand, med så mange tilfeldige kort som parameteren fortel. Dei korta som leggst i handa, skal ut av kortstokken. Metoden `taUtTilfeldig` tar ut og returnerer eit tilfeldig kort frå kortstokken. `taUtFyrste` tar ut og returnerer fyrste kort i kortstokken. Metoden `isStokka` fortel om kortstokken er stokka.

Hand: Metoden `sorter` sorterer korta på handa. Metoden `biletKort` returnerer antal biletkort, definert som alle ess, konge, dame og knekt.

Lag ferdig class Kortstokk. Vurder om dei metodane vi arvar kan nyttast som dei er, eller må reimplementerast. Hint: om vi legg eit kjent kort på botn av kortstokken, kan vi vel ikkje kalle den stokka.

Oppgåve 1d

Mange kortspel går ut på å ta stikk, ved at alle spelarane legg ut eit kort kvar, og den som la ut det høgaste kortet, får alle korta. Her må ein kunne samanlikne korta for å sjå kva for eit som er høgast. Vi kunne laga ein samanlikningsmetode i class `Kort`, men då kunne vi berre hatt *ein* samanlikningsmetode. Ulike kortspel har ulike definisjonar av kva som er høgast kort. Difor legg vi samanlikninga for seg, i ei separat klasse. Javabiblioteket har eit mønster for dette. Der finn vi fylgjande grensesnitt:

```
public interface Comparator {
    public int compare(Object o1, Object o2);
    public boolean equals(Object obj);
}
```

Klasser som implementerer dette grensesnittet må altså ha to metodar. Den fyrste, `compare`, skal kunne samanlikne to objekt som kjem inn som parametre. Om fyrste parameter er minst, skal metoden returnere eit negativt tal, t.d. -1. Om dei er like, skal den returnere 0, og dersom `o1` er størst, skal den returnere eit positivt tal, t.d. +1. `Equals`-metoden har til oppgåve å samanlikne dette samanlikningsobjektet med eit anna. Alle klasser har arva ein `equals`-metode frå class `Object`, så kravet om ein `equals`-metode er alt oppfylt.

Lag class KortSamanliknar slik at den implementerer Comparator. Du treng ikkje lage `equals`-metode. `compare` metoden skal kunne samanlikne to kort, og gje svar på kva for eit som er høgast, der korta er rangert slik: Alle spar er høgare enn alle hjerter, som er høgare enn alle ruter, som er høgare enn alle kløver. Dersom korta har same farge, er esset høgast, så konge, dame, knekt, 10, 9, ..., ned til 2 som er det lågaste.

Oppgåve 1e

Sorteringsmetoden i class `Hand` kan nytte ein ferdig sorteringsmetode i Javabiblioteket, i class `Collections` finn vi metoden

```
public static void sort(List list, Comparator c)
```

Som `Comparator` kan vi nytte eit objekt av `KortSamanliknar`-klassa. "List" på linja over er eit interface, og vi merker oss at `Vector`-klassa implementerer dette interfacet. **Lag ferdig class Hand.**

Oppgåve 2 (40%)

Du skal her lage eit enkelt kortspel for *ein* spelar. Spelet startar med at det trekkast eit tilfeldig kort. Så skal spelaren gjette på om neste kort vert høgare eller lågare *i verdi* enn dette, fargen skal ein ikkje ta omsyn til. Deretter trekkast nytt kort og samanliknast. Ess skal her tolkast som det høgaste kortet. Dersom det nye har same verdi som det gamle, skal det alltid godkjennast. Dersom spelaren gjetta feil, går ho ut av spelet utan gevinst. Dersom ho gjetta rett, kan ho velje å trekke seg med oppnådd gevinst, eller gå vidare til ny runde. For kvar runde doblast gevinsten, som startar på 100 kroner. Etter maksimalt 10 rundar kan ein vinne 51200 kroner. (Dårleg forretningsidé, spør du meg.)

Programmet skal gå nettbasert, med ein klientdel og ein tenardel, og socket-basert kommunikasjon.

Tenardelen skal halde på kortstokken og gje kort frå den. Klientdelen skal fortelle brukaren kva for eit kort som vart trukke, spørre spelaren om ho gjettar høgare, lågare, eller trekker seg. Svaret sendast tilbake til tenardelen. Det grafiske brukargrensesnittet skal ha plass for knappar til å gjette høgare, lågare, og for å trekke seg. Det skal ha plass for å vise bilete av dei korta som er trekt til no, og rundenummeret.

Tenaren skal støtte fleire samtidige klientar. Kommunikasjon mellom tenar og klient skjer etter fylgjande mønster: Etter oppkobling trekker og sender tenaren fyrste kort (send heile Kort-objektet). Så sender klienten tilbake brukarresponsen: teksten "OPP", "NED" eller "TREKKER MEG". Dersom responsen var opp eller ned, trekker tenaren nytt kort som sendast, og den ser etter om gjettinga var rett. Ved galt gjett er spelet over, og vi kan koble ned. Ved rett gjett ventar vi på neste brukarrespons. Om brukaren trekker seg, eller etter 10 runder, skal vi sende eit Integerobjekt som fortel klienten kva gevinsten vart, og avslutte.

Oppgåve 2a

Lag to figurar av brukargrensesnittet på klienten. Den eine er sjølv applikasjonen, der panela er innramma. Den andre er komponenthierarkiet, der det går fram kva for komponentar som er nytta, kva for Layoutmanagere, m.m.

Oppgåve 2b

Lag tenarapplikasjonen. Lag hjelpemetodar dersom du får lange og uoversiktlege metodar.

Lykke til!

Eksamen 852 Objektorientert Programmering 04.12.2001 - Bokmål

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

Oppgave 1 (60%)

En venn på øk/adm studiene har fått en genial forretningsidé: å la publikum spille kort på internett. Du har fått oppdraget med å utvikle dette. Det skal lages flere slags kortspill, så det første som skal lages er et generelt rammeverk for kortspill.

Faktarute : kortstokk. En kortstokk inneholder 52 vanlige kort, og eventuelt jokere. I denne oppgaven ser vi bort fra jokere. Hvert kort har en "farge" som kan være spar, hjerter, ruter eller kløver, og en "verdi" som er en av: Ess, 2, 3, 4, 5, 6, 7, 8, 9, 10, knekt, dame, konge. Sekvensen 2, 3, ..., konge er i stigende rekkefølge, men esset ser man av og til som det minste kortet (som 1), og av og til som det største (som 14). Totalt har vi $4 * 13 = 52$ ulike kort.

Du slår fast at dette rammeverket må inneholde class Kort, et objekt av denne klassen representerer ett kort i en kortstokk. Videre ser du at det er to ulike typer av samling av kort, nemlig 1) kortstokken før, under eller etter utdeling til spillerne, og 2) de "hendene" spillerne har, de kortene hver spiller sitter med i hånden. Disse to kortsamlingene ser ut til å bli ganske like. Du bestemmer deg for følgende klassehierarki:

```
public class Kortsamling { ... }
public class Kortstokk extends Kortsamling { ... }
public class Hånd extends Kortsamling { ... }
```

Etter grundig analyse, har du kommet fram til følgende skisse av class Kort og class Kortsamling:

```
public class Kort {
    public final static int KLØVER = 0;
    public final static int RUTER = 1;
    public final static int HJERTER = 2;
    public final static int SPAR = 3;
    public final static int ESS = 1;
    public final static int KNEKT = 11;
    public final static int DAME = 12;
    public final static int KONGE = 13;

    protected int farge;
    protected int verdi;

    public Kort(int farge, int verdi) { ... }
    public int getFarge() { ... }
    public int getVerdi() { ... }
    public String toString() { ... }
}
```

```

public class Kortsamling {
    protected Vector kortene;

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

```

Oppgave 1a

Lag ferdig class Kort i samsvar med skissen over. Du skal ikke introdusere nye instansvariable, det gjelder også i de andre klassene. Ved ulovlige parameterverdier i denne og andre klasser - kast gjerne en `IllegalArgumentException`. `toString`-metoden skal returnere f.eks. "Ruter 9", "Kløver ess". Forklar kort hvorfor vi ikke skal ha metodene `setFarge` og `setVerdi`.

Oppgave 1b

Lag ferdig class Kortsamling i samsvar med skissen over. Metoden `inneholder` skal returnere `true` dersom samlingen inneholder kortet med den farge og den verdi, `false` ellers. Metoden `taUt` skal ta kortet ut av vektoren og returnere kortet dersom det finnes i vektoren, kaste unntak dersom det ikke finnes. Metoden `settInn` skal sette dette kortet inn i samlingen, bakerst i vektoren. `tom` skal returnere `true` dersom samlingen er tom, `false` ellers. `antall` skal returnere antall kort i samlingen.

Oppgave 1c

Følgende skisser er laget til class `Kortstokk` og class `Hånd`:

```

public class Kortstokk extends Kortsamling {
    protected boolean stokket;

    public Kortstokk() { ... }
    public void stokke() { ... }
    public Hånd delUt(int antKort) { ... }
    public Kort taUtTilfeldig() { ... }
    public Kort taUtFørste() { ... }
    public boolean isStokket() { ... }
}

public class Hånd extends Kortsamling {
    public Hånd() { ... }
    public void sorter() { ... }
    public int bildeKort() { ... }
}

```

Kortstokk: Instansvariablen `stokket` skal holde orden på om kortene i kortstokken er stokket, dvs. om de ligger i tilfeldig rekkefølge. Konstruktøren skal initialisere instansvariablene, herunder lage de 52 kortene (ved hjelp av løkker, ikke 52 setninger). `stokke` skal stokke om de kortene som finnes slik at de ligger i tilfeldig rekkefølge. Opprett

gjør en ekstra vektor og flytter kortene over dit i tilfeldig rekkefølge ved hjelp av en random-funksjon. `delUt` skal lage og returnere en hånd, med så mange tilfeldige kort som parameteren forteller. De kortene som legges i hånden, skal ut av kortstokken. Metoden `taUtTilfeldig` tar ut og returnerer et tilfeldig kort fra kortstokken. `taUtFørste` tar ut og returnerer første kort i kortstokken. Metoden `isStokket` forteller om kortstokken er stokket.

Hånd: Metoden `sorter` sorterer kortene på hånden. Metoden `bildeKort` returnerer antall bildekort, definert som alle ess, konge, dame og knekt.

Lag ferdig class Kortstokk. Vurder om de metodene vi arver kan benyttes som de er, eller må reimplementeres. Hint: om vi legger et kjent kort på bunnen av kortstokken, kan vi vel ikke kalle den stokket.

Oppgave 1d

Mange kortspill går ut på å ta stikk, ved at alle spillerne legger ut et kort hver, og den som la ut det høyeste kortet, får alle kortene. Her må man kunne sammenligne kortene for å se hvilket som er høyest. Vi kunne lage en sammenligningsmetode i class `Kort`, men da kunne vi bare hatt *en* sammenligningsmetode. Ulike kortspill har ulike definisjoner av hva som er høyeste kort. Derfor legger vi sammenligningen for seg, i en separat klasse. Javabiblioteket har et mønster for dette. Der finner vi følgende grensesnitt:

```
public interface Comparator {
    public int compare(Object o1, Object o2);
    public boolean equals(Object obj);
}
```

Klasser som implementerer dette grensesnittet må altså ha to metoder. Den første, `compare`, skal kunne sammenligne to objekter som kommer inn som parametre. Dersom første parameter er minst, skal metoden returnere et negativt tall, f.eks. -1. Om de er like, skal den returnere 0, og dersom `o1` er størst, skal den returnere et positivt tall, f.eks. +1. `Equals`-metoden har til oppgave å sammenligne dette sammenligningsobjektet med et annet. Alle klasser har arvet en `equals`-metode fra class `Object`, så kravet om en `equals`-metode er alt oppfylt.

Lag class KortSammenligner slik at den implementerer Comparator. Du trenger ikke lage `equals`-metode. `compare` metoden skal kunne sammenligne to kort, og gi svar på hvilket som er høyest, der kortene er rangert slik: Alle spar er høyere enn alle hjerter, som er høyere enn alle ruter, som er høyere enn alle kløver. Dersom kortene har samme farge, er esset høyest, så konge, dame, knekt, 10, 9, ..., ned til 2 som er det laveste.

Oppgave 1e

Sorteringsmetoden i class `Hånd` kan benytte en ferdig sorteringsmetode i Javabiblioteket, i class `Collections` finner vi metoden

```
public static void sort(List list, Comparator c)
```

Som `Comparator` kan vi benytte et objekt av `KortSammenligner`-klassen. "List" på linjen over er et interface, og vi merker oss at `Vector`-klassen implementerer dette interfacet. **Lag ferdig class Hånd.**

Oppgave 2 (40%)

Du skal her lage et enkelt kortspill for *en* spiller. Spillet starter med at det trekkes et tilfeldig kort. 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 nytt kort og sammenlignes. Ess skal her tolkes som det høyeste kortet. Dersom det nye 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 trekke seg med oppnådd gevinst, eller gå videre til ny runde. For hver runde dobles gevinsten, som starter på 100 kroner. Etter maksimalt 10 runder kan man vinne 51200 kroner. (Dårlig forretningsidé, spør du meg.)

Programmet skal gå nettbasert, med en klientdel og en tjenerdel, og socket-basert kommunikasjon.

Tjenerdelen skal holde på kortstokken og gi kort fra den. Klientdelen skal fortelle brukeren hvilket kort som ble trukket, spørre spilleren om hun gjetter høyere, lavere, eller trekker seg. Svaret sendes tilbake til tjenerdelen. Det grafiske brukergrensesnittet skal ha plass for knapper til å gjette høyere, lavere, og for å trekke seg. Det skal ha plass for å vise bilde av de kortene som er trukket til nå, og rundenummeret.

Tjeneren skal støtte flere samtidige klienter. Kommunikasjon mellom tjener og klient skjer etter følgende mønster: Etter oppkobling trekker og sender tjeneren første kort (send hele Kort-objektet). Så sender klienten tilbake brukerresponsen: teksten "OPP", "NED" eller "TREKKER MEG". Dersom responsen var opp eller ned, trekker tjeneren nytt kort som sendes, og den ser etter om gjettingen var rett. Ved galt gjett er spillet over, og vi kan koble ned. Ved rett gjett venter vi på neste brukerrespons. Om brukeren trekker seg, eller etter 10 runder, skal vi sende et Integerobjekt som forteller klienten hva gevinsten ble, og avslutte.

Oppgave 2a

Lag to figurer av brukergrensesnittet på klienten. Den ene er selve applikasjonen, der panelene er innrammet. Den andre er komponenthierarkiet, der det går fram hvilke komponenter som er benyttet, hvilke Layoutmanagere, o.l.

Oppgave 2b

Lag tjenerapplikasjonen. Lag hjelpemetoder dersom du får lange og uoversiktlige metoder.

Lykke til!