NTNU
Norges teknisk-
naturvitenskapelige universitet

Fakultet for informasjonsteknologi,
matematikk og elektroteknikk
Institutt for datateknikk
og informasjonsvitenskap

BOKMÅL

EKSAMEN I FAG
TDT4100 Objektorientert programmering

Fredag 6. juni 2008
Kl. 09.00 – 13.00

Faglig kontakt under eksamen:
Hallvard Trætteberg, tlf (735)93443 / 918 97263

Tillatte hjelpemidler:

- Én og kun én trykt bok, f.eks. Liang: Introduction to Java Programming.

Sensur:
Sensuren faller 3 uker etter eksamen og gjøres deretter kjent på http://studweb.ntnu.no/.

Prosentsatser viser hvor mye hver oppgave teller innen settet.

Merk: All programmering skal foregå i Java.

Lykke til!

OPPGAVE 1 (40%): Sjakkbrikker- og brett

Du skal lage et sjakkspill, med klasser for å representere brikkefarge, brikker og brettet med alle brikkene. Reglene er noe forenklet, bl.a. er omgjøring av bønder og rokkade utelatt.

Sjakk spilles på et 8x8 brett, hvor rutene angis med bokstav a-h for kolonnen og tall 1-8 for raden. Nederste venstre hjørne har koordinatene ”a1” og motsatt hjørne har koordinatene ”h8”. En rute på brettet kan være tom eller inneholde en hvit eller sort brikke.

Det er 6 forskjellige typer brikker, med hver sine regler for hvordan de flytter. Generelt kan en brikke enten flytte til et tomt felt eller til et felt med en brikke med motsatt farge, heretter kalt en motstanderbrikke. Det siste kalles å slå og brikken som flyttes vil da erstatte motstanderbrikken.

- *Bonden* flytter normalt 1 rute frem, men kan også flytte 2 (uten å hoppe over andre brikker) dersom den (fortsatt) står i utgangsposisjonen. Bonden slår motstandere 1 rute diagonalt forover, dvs. oppover for hvit og nedover for svart.

- *Springeren* (hest) flytter 2 ruter i én retning og 1 rute i retning vinkelrett på første, f.eks. 2 ruter frem og 1 til venstre eller 1 rute bakover og 2 til høyre. Springeren kan hoppe over andre brikker.
- *Løperen* (biskop) flytter 1 eller flere ruter diagonalt i en av 4 retninger og kan ikke hoppe over andre brikker.
- *Tårnet* flytter 1 eller flere ruter rett frem eller til siden i en av 4 retninger og kan ikke hoppe over andre brikker.
- *Dronningen* flytter 1 eller flere ruter rett frem, til siden eller diagonalt i en av 8 retninger og kan ikke hoppe over andre brikker.
- *Kongen* flytter 1 rute rett frem, til siden eller diagonalt i en av 8 retninger. Dersom kongen står slik at den kan slås av motstanderen i neste trekk, så er den *sjakk*.

.

Ved start har hver spiller 8 bønder, 2 springere, 2 løpere, 2 tårn, 1 dronning og 1 konge. Den ene spilleren har hvite brikker og den andre svarte. Utgangsposisjonen er som vist i figuren, med 1 rad bønder og 1 rad med hhv. tårn, springer, løper, dronning, konge, løper, springer og tårn, for hver spiller. De hvite bøndene flytter oppover og de sorte bøndene nedover.

Nedenfor vil vi beskrive det du skal implementere i mer detalj. *Dersom du ikke klarer å implementere en eller flere metoder, så husk at du likevel kan bruke dem i andre oppgaver, slik at du viser hva du kan!* Innfør gjerne hjelpemetoder for å gjøre løsningen ryddigere.

a) Definer en enum-klasse kalt **PieceColor**, med verdiene **WHITE** og **BLACK**. Implementer metoden **getOtherColor**, som ikke tar noen parametre og som returnerer den andre fargeverdien. Dersom du ikke vet hvordan enum-klassen og metoden implementeres, så beskriv en alternativ teknikk for å representere fargen til en brikke. Hvorfor er en enum-klasse å foretrekke fremfor å bruke f.eks. **String** eller **int**?

Grensesnittet **Piece** er definert som følger:

```
public interface Piece {
      public PieceColor getPieceColor();
      public boolean canTake(String from, String to, Board board);
      public boolean canMove(String from, String to, Board board);
}
```

Tanken er at dette grensesnittet skal implementeres av 6 brikkeklasser **Pawn** (bonde), **Knight** (springer), **Bishop** (løper), **Rook** (tårn), **Queen** (dronning) og **King** (konge). Merk at det kun er tre av disse som skal implementeres i en av del-oppgavene nedenfor.

Klassen **Board** skal representere sjakkbrettet vha. en tabell (array) med **Piece**-objekter, med dimensjoner 8x8. Metodene som refererer til ruter, skal bruke **String**-posisjoner (koordinater) på formatet ”a1” til ”h8”, som vist i figuren over.

b) Implementer **Board**-klassen med den interne tabellen og følgende to *innkapslingsmetoder*.

```
Piece getPiece(String position) { … }
```

```
    void setPiece(String position, Piece piece) { … }
```

- **getPiece** returnerer brikken på ruta angitt med **position,** eller null dersom ruta er tom. F.eks. skal **getPiece(”a1”)** og **getPiece(”h8”)** returnere en evt. brikke i hhv. ruta nederst til venstre og øverst til høyre.
- **setPiece** plasserer brikken **piece** i ruta angitt med **position**.

Hint: Definer hjelpemetoder for å konvertere til/fra **String**-posisjoner og indeksene til tabellen, f.eks. **getPositionRow(String)** og **getPositionColumn(String)** og **getPosition(int,int)**, og bruk disse i løsningen i denne og senere deloppgaver. Dersom du er usikker på hvordan disse implementeres, så definer presist hva de gjør og bruk dem likevel.

c) Implementer følgende hjelpemetoder i **Board**-klassen (disse metoden er nyttige når logikken for lovlige flytt skal implementeres i brikkeklassene):

```
    boolean isStraight(String from, String to) { … }
    boolean isDiagonal(String from, String to) { … }
    boolean isOccupiedBetween(String from, String to) { … }
```

- **isStraight** returnerer en **boolean** som angir om et flytt fra **from** til **to** går *langs* en rad eller kolonne, altså i en av de 4 retningene parallelt med kantene på brettet.
- **isDiagonal** returnerer en **boolean** som angir om et flytt fra **from** til **to** går diagonalt, altså i en av de 4 retningene på skrå over brettet. Hint: Hva er forholdet mellom endringen i rad og kolonne i et diagonalt flytt?
- **isOccupiedBetween** skal returnere en **boolean** som angir om én eller flere av rutene mellom (og ikke inkludert) **from** og **to** er fylt med en brikke.

d) Hva betyr modifikatoren **static**? Slike hjelpemetoder som **isStraight**, **isDiagonal** og **isOccupiedBetween** har ofte **static**-modifikatoren, hvorfor?

e) Implementer **Rook** (tårn)-, **Queen** (dronning)- og **Knight**-klassene, som alle implementerer grensesnittet **Piece**. Hva er fordelen med å definere et slikt felles grensesnitt? Implementer også en konstruktør, som brukes for å sette brikkefargen. Logikken til metodene i **Piece**-grensesnittet skal være som følger:

- **getPieceColor** returnerer fargen til brikken.
- **canMove** returnerer en **boolean** som angir om denne brikken, dersom den står på **from**-posisjonen, har lov til å flytte til **to**-posisjonen. Du trenger ikke sjekke om brikken faktisk står på **from** eller om **to** er tom.
- **canTake** returnerer en **boolean** som angir om denne brikken, dersom den står på **from**-posisjonen, har lov til å ta en motstanderbrikke på **to**-posisjonen. Du trenger ikke sjekke om brikken faktisk står på **from** eller om det faktisk står en motstanderbrikke på **to**.

OPPGAVE 2 (25%): Sjakk forts.

a) Du har kanskje oppdaget at brikkeklassene har en del kode som er lik. Beskriv hvordan en abstrakt superklasse, f.eks. kalt **AbstractPiece**, kan innføres og implementeres for å unngå duplisering av kode (om du ikke har implementert brikkeklassene, så svar generelt på spørsmålet). Her må du ta høyde for implementasjonen av de andre brikke-klassene, f.eks. **Pawn** (bonde). Hvorfor bør en slik superklasse være abstrakt?

b) Forklar hvordan du kan sikre at en **Board**-instans har en gyldig start-tilstand og at den ikke kan settes i en ugyldig tilstand utenifra (fra andre klasser), dvs. kun lovlige trekk kan utføres?

c) Definer en unntaksklasse for ulovlige posisjoner, f.eks. ”a0”, ”i3” og ”tdt4100”, og forklar hvordan den brukes for å si fra om slike feil. Begrunn valg av superklasse.

d) Anta at **Board** har en konstruktør som setter opp brikkene riktig og at klassen i tillegg har følgende metoder:

```
boolean isLegalMove(PieceColor color, String from, String to) { … }
void movePiece(String from, String to) { … }
boolean isCheck(PieceColor color) { … }
```

- **isLegalMove** returnerer en **boolean** som angir om det er lov for spilleren med fargen **color** å flytte brikken på **from**- posisjonen til **to**-posisjonen.
- **movePiece** flytter en brikke fra **from**-posisjonen til **to**-posisjonen.
- **isCheck** returnerer en **boolean** som angir om kongebrikken med fargen **color** er sjakk.

Implementer en **main**-metode, som oppretter et sjakkbrett og lar brukeren skrive inn flytt (velg selv formatet). Husk på at hvit trekker først og at de deretter veksler mellom å trekke. Programmet skal si fra når en konge blir sjakk og avslutte når en konge blir tatt. Ellers trenger du ikke bry deg om regler ut over de som er beskrevet over.

Til høyre ser du mulig utskrift og input ved kjøring av en slik main-metode.

```
WHITE's turn:
        d2-d4
BLACK's turn:
        e7-e5
WHITE's turn:
        d2-d4
Illegal move!
WHITE's turn:
        d4-e5
BLACK's turn:
```

OPPGAVE 3 (20%): Observert-observatør-samspill

Anta at du har laget **Board**-klassen over, bl.a. med metoder for å plassere/flytte brikkene på brettet. Forklar med tekst og evt. kodesnutter hvordan du vil gå frem for å gjøre **Board** ”observerbar”, dvs. gjøre det mulig for (instanser av) andre klasser å ”følge med” spillets gang. Hvilke klasser, grensesnitt og metoder vil du innføre og hvordan må de virke?

OPPGAVE 4 (15%): Iterator

a) Anta at det finnes en **Iterator<String>**-implementasjon kalt **BoardIterator** som ”generer” alle **String**-posisjoner ”a1” til ”h8” på brettet (i en eller annen rekkefølge). Bruk **BoardIterator** til å implementere en metode String findKind(PieceColor pieceColor), som returnerer posisjonen til **King**-brikken med den angitte **pieceColor**.

b) Implementer **BoardIterator**-klassen. Hint: Husk at settet med posisjoner er fast (statisk).

c) Hva må en gjøre med **Board**-klassen for å kunne skrive **for (String position: board) { ... }** (en såkalt enhanced for-loop eller foreach-loop), dersom **board** er deklarert som **Board**?