

# I110 Høsten 2001, obligatorisk oppgave 2

## Krav til innlevering

Den obligatoriske oppgaven skal leveres skriftlig.

For at denne obligatoriske oppgaven skal bli godkjent må følgende leveres:

- Oversikt

Et dokument som inneholder en oversikt over innleveringen. Den skal inneholde en kort beskrivelse av alle filene som blir levert inn og skal også oppgi navn, gruppe og email-adresse. Hvis man ikke går på noen gruppe bør man skrive navnet til gruppelederen som har rettet den første obligatoriske oppgaven. Dette dokumentet er viktig siden det hjelper den som skal rette oppgaven å sette seg inn i besvarelsen.

- Dokumentasjon

Det skal legges ved dokumentasjon som beskriver utformingen av programmet (f.eks. klasse-diagram). Denne dokumentasjonen bør være så god at en som ikke er kjent med programmet eller oppgaven kan forstå rollen til hver klasse og hvordan hver metode fungerer uten å være nødt til å lese selve kildekoden.

- Pseudokode

Som en del av dokumentasjonen kan det være nyttig å legge ved pseudokode som gir en skisse over hvordan programmet fungerer.

- Kildekode

All kildekoden skal selvsagt legges ved. All kildekode **skal** inneholde javadoc-kommentarer til klasser og metoder (for et eksempel, se kommentarene i interface-koden).

- Kjøre-eksempel

En fil som inneholder en logg av en typisk kjøring av programmet. Et slikt kjøre-eksempel viser hvordan programmet brukes og hva programmet skriver ut.

- Kjørbar versjon

For at den som retter oppgaven skal kunne sjekke at programmet fungerer som det skal, må en kjørbart versjon legges åpen. Angi i innleveringen hvilken fil det dreier seg om, og åpne filen ved kjøre kommandoen:

```
chmod a+r [filnavn]
```

Ligger filene i en underkatalog må også denne åpnes. Dette gjøres ved å kjøre kommandoen:

```
chmod a+rx [katalognavn]
```

- Innleveringsfrist

Innleveringsfristen er 2. november 2001, Kl 15:00. Innlevering etter fristen vil resultere i "ikke godkjent" oblig. Dersom man leverer innen fristen men ikke får godkjent, vil man få en sjans til å levere en forbedret utgave. Detaljer om hvordan innleveringen skal skje vil bli beskrevet på I110-websiden.

## Spillet Fire-På-Rad (FPR)

FPR spilles på et rektangulært brett. Det er to spillere som veksler på å gjøre sine trekk. En spiller har røde brikker og den andre har blå brikker. Det rektangulære brettet er delt inn i *felt*. For eksempel vil et 6 x 7 brett ha 42 felt. Se for deg at brettet står på høykant. Brikker slippes *vertikalt* ned i *kolonnene* av felt. En brikke som slippes ned i en kolonne faller til bunnen av kolonnen dersom kolonnen er tom, eller lander på toppen av brikkene som allerede er i denne kolonnen. I standard versjonen av spillet vinner den første av spillerene som har fire av sine brikker på rad, enten i en kolonne, rekke eller diagonal.

Du skal lage et program som implementerer spillet FPR for to spillere. Brettets størrelse skal være valgfri (men minst 4 x 4), og skal kunne velges ved start av et nytt spill. Programmet skal tilby brukere et tekstbasert brukergrensesnitt til spillet, der det opplyses om spillets tilstand og spør etter neste trekk fra den spilleren som har sin tur. Dersom spilleren prøver å gjøre trekk som ikke er gyldig, som f.eks. å velge et ugyldig kolonnennummer eller en kolonne som er full, skal programmet gi en feilmelding og be spilleren om å gjøre et gyldig trekk. Ved oppstart av programmet skal brukeren av programmet få en del valg:

- Hvor mange rekker spillebrettet skal ha. Minste tillatte antall er 4.
- Hvor mange kolonner spillebrettet skal ha. Minste tillatte antall er også her 4.
- Hvilken spiller (1 eller 2) som skal begynne spillet.

Under spillets gang skal programmet for hver runde (et trekk):

- Skrive ut spillebrettets tilstand (tekstrepresentasjon)
- Be den spilleren som skal gjøre neste trekk om et trekk.
- Dersom trekket ikke var gyldig, skrive ut feilmelding og be om et nytt trekk.
- Dersom trekket førte til en vinner, skrive en beskjed om dette og avslutte spillet.

Ved kjøring av programmet kan det f. eks. se slik ut:

Spillebrettets (5x7) tilstand er:

0 1 2 3 4 5 6 (kolonnenr.)

```
b
r b      r
r b b    b r
r r r    b b r
```

Hva er ditt neste trekk, spiller 2 (blå)? 8

8 er et ugyldig trekk! Velg et annet trekk: 3

Spillebrettet (5x7) tilstand er:

0 1 2 3 4 5 6 (kolonnenr.)

```
b
r b      r
r b b    b r
r r r b b b r
```

Gratulerer, spiller nummer 2 (blå), du har vunnet!

Alternative utforminger av brukergrensesnittet aksepteres. Eneste krav til brukergrensesnittet er at det skal være lett forståelig for brukeren av programmet.

Programmet skal implementeres i klasser som representerer spillet, spillebrettet, brikker og spillere. Klassene du skal lage **skal** implementere et sett med kontrakter (interfaces) som er gitt nedenfor (IBrikke, ISpiller, ISpilleBrett, ISpill, se slutten av oppgaveteksten). Disse kontraktene kan du laste ned fra I110-oppgavesiden. Det tekstbaserte brukergrensesnittet skal implementeres i en egen klasse. Denne klassen skal inneholde minst mulig informasjon som har med spillets tilstand gjre. Informasjonen om spillets tilstand skal i hovedsak hndteres av klassene som implementerer kontraktene (interfacene) IBrikke, ISpiller, ISpilleBrett, og ISpill. Programmet må holde rede på følgende informasjon:

- Spillebrettets størrelse (antall rekker og kolonner).
- Spillebrettets tilstand (brikker som er på spillebrettet).
- Hvilken spiller det er sin tur til å gjøre neste trekk.
- Om noen av spillerene har vunnet (spillet er ferdig), og i tilfelle hvem som vant.
- Spillere og spillerfarger.

## Kontrakter som skal implementeres

Disse kontraktene kan du laste ned fra I110 oppgavesiden (filen Oblig2Kontrakter.java). Legg filen i samme katalog som resten av programfilene dine og kompiler den.

```
import java.awt.*;

/**
 * Kontrakt for implementasjon av en spiller
 */

interface ISpiller{

    /**
     * @return int Nummeret til spilleren (1 eller 2)
     */
    int hentSpillerNummer();

    /**
     * @return Color Spillerens farge (Color.red eller Color.blue)
     */
    Color hentSpillerFarge();
}

/**
 * Kontrakt for implementasjon av en spillebrikke
 */

interface IBrikke{

    /**
     * @return int Rekken til brikken på spillebrettet
```

```
*/  
int hentRekke();
```

```

/**
 * @return int Kolonnen til brikken på spillebrettet
 */
int hentKolonne();

/**
 * @return ISpiller Spilleren som gjorde trekket med denne brikken
 */
ISpiller hentSpiller();
}

/**
 * Kontrakt for implementasjon av et Fire-På-Rad spillebrett
 */

interface ISpilleBrett{

    /**
     * @return IBrikke[] [] Spillebrettet som en IBrikke array
     */
    IBrikke [] [] hentSpilleBrett();

    /**
     * @return boolean True dersom den spesifiserte kolonnen ikke er full,
     * ellers false (kan ikke gjøre et trekk i denne kolonnen)
     */
    boolean gyldigKolonneNummer(int kolonne);

    /**
     * Setter inn en brikke på brettet for den spesifiserte spillere og
     * kolonne. Fargen på brikken bestemmes av spillerens farge.
     * @return IBrikke Brikken som ble lagt til brettet.
     */
    IBrikke leggTilBrikke(int kolonne, ISpiller spiller);

    /**
     * @return int Antall kolonner på spillebrettet
     */
    int hentAntallKolonner();

    /**
     * @return int Antall rekker på spillebrettet
     */
    int hentAntallRekker();
}

```

```

/**
 * Kontrakt for implementasjon av spillet Fire På Rad
 */

interface ISpill{

    /**
     * Gjør et trekk i den spesifikke kolonnen for den spesifiserte spilleren
     * @return int Rekkenummeret som trekket ble gjort i, ellers verdi < 0
     */
    int gjørTrek(int kolonne, int spillerNummer);

    /**
     * @return ISpiller Spilleren med det spesifiserte nummeret
     */
    ISpiller hentSpiller(int spillerNummer);

    /**
     * @return ISpiller Spilleren som gjorde det siste trekket
     */
    ISpiller hentNåværendeSpiller();

    /**
     * @return ISpiller Spilleren som vant spillet, dersom spillet er ferdig,
     * ellers null
     */
    ISpiller hentVinner();

    /**
     * @return ISpilleBrett Spillebrettet som spillet spilles på
     */
    ISpilleBrett hentSpilleBrett();

    /**
     * @return boolean True dersom spillet er ferdig, ellers false
     */
    boolean spillErFerdig();
}

```