

Omkontrollskrivning, Programmeringsteknik för D/C

2012-01-10, 8.00-13.00

Anvisningar: Preliminärt ger uppgifterna $23 + 7 + 10 = 40$ poäng. För godkänt betyg krävs ungefär 25 poäng. Tillåtet hjälpmedel: Java-snabbreferens.

Betyget på denna skrivning utgör betyget på kontrollskrivningarna, oavsett resultatet på tidigare skrivningar (dock sänks inte ett tidigare godkänt betyg). Resultatlistan anslås på institutionens anslagstavla när rättningen är klar.

I Vattenhallen (Science Center LTH) har LTHs studenter precis byggt upp en stor spelplan för det klassiska spelet *Minesweeper* (*MS Røj* eller *Minrøj* på svenska) för en tävling mellan olika universitet i Sverige. Uppgift 1 och 2 i denna tentamen handlar om klasser för en förenklad version av originalspelet. Om du har spelat spelet någon gång så tänk på att det är en förenklad version av spelet som ska implementeras här!

1. En *förenklad* version av spelet Minesweeper (MS Røj eller Minrøj på svenska) har följande beskrivning:

Spelplanen består av ett kvadratisk rutnät där ett känt antal minor är gömda, och spelet går ut på att markera alla rutorna som innehåller minor.

Från början är alla rutorna dolda (oöppnade), och i varje drag kan man antingen öppna eller markera en av de kvarvarande orörda rutorna:

- Man öppnar en ruta genom att vänsterklicka på den. Om rutan innehåller en mina är spelet förlorat. Om rutan inte innehåller en mina visas en siffra 0..8, som anger antalet minor i grannrutorna till den öppnade rutan.
- Man markerar en oöppnad ruta genom att högerklicka på den. På så sätt anger man att man tror att rutan innehåller en mina.
- Har man öppnat eller markerat en ruta kan man inte ändra den igen.

Man väljer själv när man vill avsluta spelet (om det inte avslutas genom att man försökt öppna en ruta med en mina). Om man då markerat alla rutor med minor, och inga andra rutor, har man vunnit spelet. Man behöver alltså inte ha öppnat alla övriga rutor.

Här är några exempel på hur en spelplan med 4×4 rutor kan se ut. Spelplanen innehåller tre minor.

? ? ? ?	1 1 1 ?	1 1 1 ?	1 1 1 0
? ? ? ?	1 M 2 ?	1 M 2 ?	1 M 2 1
? ? ? ?	? 2 M ?	? 2 M ?	1 2 M 2
? ? ? ?	0 ? ? ?	0 ? ? X	0 1 2 M
Startspelplanen	2 korrekt markerade rutor	Förlust (mina öppnad)	Vinst

? är en oöppnad och omarkerad ruta (orörd ruta).

M är en oöppnad men markerad ruta.

X är en öppnad ruta med en mina.

0..8 är en öppnad ruta där siffran visar hur många av grannrutorna som innehåller minor.

Klassen MinesweeperSquare beskriver en ruta på spelplanen. Klassen är färdigskriven (du ska alltså *inte* implementera den).

MinesweeperSquare

```
/** Skapar en oöppnad och omarkerad ruta som inte innehåller någon mina. */
MinesweeperSquare();

/** Placerar en mina i rutan. */
void putMine();

/** Markerar rutan (om oöppnad och omarkerad, annars händer ingenting). */
void mark();

/** Öppnar rutan (om oöppnad och omarkerad, annars händer ingenting). */
void open();

/** Returnerar true om rutan är öppnad, false annars. */
boolean isOpened();

/** Returnerar true om rutan är markerad, false annars. */
boolean isMarked();

/** Returnerar true om rutan innehåller en mina, false annars. */
boolean isMined();
```

(fortsätter nästa sida)

Klassen MinesweeperBoard beskriver en hel spelplan.

MinesweeperBoard

```
/** Skapar en spelplan av storlek size*size med mines minor slumpmässigt
    utplacerade på spelplanen. */
MinesweeperBoard(int size, int mines);

/** Öppnar rutan (row,col) om möjligt. Returnerar false
    om (row,col) är en minruta, true annars. */
boolean open(int row, int col);

/** Markerar rutan (row,col) om möjligt. */
void mark(int row, int col);

/** Returnerar true om alla rutor med minor är markerade och endast
    dessa, annars false. */
boolean win();

/** Skriver ut aktuellt utseende av hela spelplanen. Varje ruta
    beskrivs av ett tecken enligt nedan.
    En öppnad och omarkerad ruta beskrivs med "?".
    En markerad (öppnad) ruta beskrivs med "M".
    En öppnad ruta med en mina beskrivs med "X".
    En öppnad ruta utan mina beskrivs med ett heltal (0..8) som visar
    antal minor i grannrutorna. */
void print();
```

Implementera klassen MinesweeperBoard.

Anvisningar:

- Raderna och kolumnerna på spelplanen numreras från 1 och uppåt.
- Du får förutsätta att row och col vid anrop alltid har värden inom spelplanen.
- Du ska använda en matris med element av typen MinesweeperSquare för spelplanen.
- För att underlätta räkningen av antalet minor i grannrutorna är det lämpligt att använda en ram runt matrisen (dvs göra matrisen ett steg större i alla riktningar).
- Det underlättar också att lägga koden för att räkna antalet minor i grannrutorna i en metod. Därför ska du implementera (och använda) följande privata metod för detta:

```
/** Returnerar antal minor i grannrutorna (0..8) till rutan (row,col). */
private int getMinesAround(int row, int col) {
    ...
}
```

- Utskriften i print-metoden ska vara enligt de fyra exemplen på sidan 2.

2. Implementera ett huvudprogram som genomför en omgång av spelet.

Anvisningar:

- Storleken på spelplanen och antalet minor ska läsas in i början av programmet.
- Rita upp spelplanen efter varje utfört drag.
- För varje drag: läs in tre värden som simulerar var och hur man klickat på spelplanen. Det första värdet anger hur man klickat (V = vänsterklick, H = högerklick), medan de två sista värdena anger rad och kolumn. För att avsluta spelet (och kontrollera om man vunnit) matar man bara in något annat än V eller H.
Exempel: V 3 2 betyder att man vänsterklickat på ruta (3,2).
- Avsluta programmet med att skriva ut om man vunnit eller förlorat.

Du ska naturligtvis använda klasserna i uppgift 1 för att genomföra spelet.

3. Implementera följande tre metoder (de är tänkta att finnas i en klass med bara statiska metoder som arbetar med listor av strängar):

```
/** Returnerar en lista med alla strängar i list som inleds med  
    tecknet ch. */  
static ArrayList<String> startsWith(ArrayList<String> list, char ch);
```

```
/** Ändrar ordningen på elementen i list så de kommer i omvänd  
    ordning (första elementet sist, andra elementet näst sist, osv). */  
static void reverse(ArrayList<String> list);
```

```
/** Returnerar en lista med alla strängarna i list men i längdordning  
    med de längsta strängarna först. Ordningen mellan strängar av  
    samma längd spelar ingen roll. */  
static ArrayList<String> lengthOrder(ArrayList<String> list);
```

Anvisningar:

- Du får förutsätta att list inte är null i alla tre metoderna, men de ingående strängarna kan vara tomma (dvs ha längden noll).
- För full poäng får inte ursprungslistan list förstöras i metoderna startsWith och lengthOrder.
- Det finns en klass Collections i Java som bland annat har metoder som gör vad du ska göra – du får naturligtvis inte använda dessa metoder :-)