Laboration 8 – matriser, Memory-spel

Mål: Du ska träna på att att använda matriser.

Förberedelseuppgifter

- Läs avsnittet Bakgrund.

Bakgrund

Memory är ett spel som går ut på att hitta matchande par på en spelplan med kort. I spelet används ett jämnt antal kort, där det finns exakt två kort med samma motiv på framsidan.

Spelet går till som följer. Alla korten blandas och placeras ut på spelplanen med baksidan uppåt. Spelaren vänder på två kort i taget. Om de båda korten visar olika bilder måste de vändas upp och ner igen. Om de visar samma bild får de ligga kvar med framsidan upp. Spelet fortsätter tills spelaren lyckats hitta alla par.

Datorarbete

1. För att beskriva en (tvåsidig) bild av ett memorykort används den färdiga klassen `MemoryCardImage`.

   `MemoryCardImage`

   ```
   /** Skapar en tvåsidig bild av ett memorykort.
       Bilden på framsidan finns i filen med namnet frontFileName och
       bilden på baksidan i filen med namnet backFileName. */
   MemoryCardImage(String frontFileName, String backFileName);

   /** Returnerar bilden på framsidan. */
   Image getFront();

   /** Returnerar bilden på baksidan. */
   Image getBack();
   ```

   Klassen `MemoryCardImage` finns i ditt projekt *Lab08*. Titta på koden, se vilka metoder som finns och ungefär hur de är skrivna. Kan du förstå koden på ett ungefär?

2. Klassen `MemoryBoard` beskriver ett bräde med 16 memorykort. Ett ”skelett” till klassen finns i projektet *Lab08*. Din uppgift är att implementera den enligt specifikationen nedan.

   `MemoryBoard`

   ```
   /** Skapar ett memorybräde med size * size kort.
       backFileName är filnamnet för filen med baksidesbilden.
       Vektorn frontFileNames innehåller filnamnen för frontbilderna. */
   MemoryBoard(int size, String backFileName, String[] frontFileNames);

   /** Tar reda på brädets storlek. */
   int getSize();

   /** Hämtar den tvåsidiga bilden av kortet på rad r, kolonn c.
   Raderna och kolonnerna numreras från 0 och uppåt. */
   MemoryCardImage getCard(int r, int c);

   /** Vänder kortet på rad r, kolonn c. */
   void turnCard(int r, int c);
   ```

```
/** Returnerar true om kortet r, c har framsidan upp. */
boolean frontUp(int r, int c);

/** Returnerar true om det är samma kort på rad r1, kolonn c1
    som på rad r2, kolonn c2. */
boolean same(int r1, int c1, int r2, int c2);

/** Returnerar true om alla kort har framsidan upp. */
boolean hasWon();
```

Att placera ut korten

När brädet skapas ska 8 st `MemoryCardImage`-objekt skapas. Varje sådant objekt ska placeras ut på två slumpmässiga platser på brädet. Om det redan finns ett objekt på en vald plats måste du välja en ny plats. Du kan alltså behöva dra slumptal flera gånger för att placera ut ett kort. En sådan algoritm kan informellt formuleras så här:

Upprepa följande för vart och ett av filnamnen i `frontFileNames`:

- Skapa ett `MemoryCardImage`-objekt med detta namn.
- Placera **två** referenser till detta objekt på brädet.
  Varje referens placeras ut så här:
  - Gissa (dra slumptal) värden på rad och kolumn för kortet.
    Kalla dessa värden r och c.
  - Så länge platsen (r, c) är upptagen på brädet, dra nya slumptal för r och c.
  - Därefter vet vi att platsen (r, c) är ledig.
    Det går därför bra att stoppa in en ny referens där.

Ledning

- Du måste alltså skilja på den bild av spelet som den som kör programmet har och hur det ser ut inuti klassen `MemoryBoard`. Spelaren ser 16 kort framför sig, men i själva verket representeras korten istället av de 8 `MemoryCardImage`-objekten.
- För att hålla reda på om ett kort har framsidan upp eller ej ska du använda en `boolean`-matris med samma dimensioner som brädet.

  Klassen `MemoryWindow` anropar metoden `frontUp` i `MemoryBoard` för att avgöra om ett givet kort ska ritas som upp- eller nedvänt. Det huvudprogram, som du kommer att skriva senare i laborationen, kommer också att använda den metoden för att avgöra om ett kort kan vändas upp eller ej.

  Medan du testar utplaceringen av kort i din `MemoryBoard`-klass (algoritmen ovan) kan det vara praktiskt att låta alla kort ritas uppvända från början. Då kan vi se att utplaceringen av korten fungerar som den ska. För sådan testning ska alltså metoden `frontUp` returnera `true` för alla platser på brädet.
- Vi vill gärna undvika alltför långa och stökiga metoder, och detta gäller särskilt konstruktorer. I kodskelettet finns därför en privat hjälpmetod `createCards`. Privata metoder syns inte i specifikationen, men genom att implementera metoden och använda den från konstruktorn blir konstruktorn kortare och koden mer lättläst.

  Det är inte nödvändigt att använda just denna privata metod: välj gärna en egen.

- När man utvecklar program implementerar man oftast inte en hel klass i taget, som det beskrivs här, utan det normala är att man vill testa mycket och ofta. De första testerna gör man helst innan man har hunnit skriva allt för mycket kod.

  För denna uppgift är det rekommenderat att implementera konstruktorn och metoderna `getSize` samt `getCard` och därefter gå vidare till `main`-metoden i `MemoryGame` (se uppgifterna 3 och 4) och där kontrollera att man kan skapa och rita upp ett bräde med alla kortens baksidor upp. När det fungerar implementerar man resten av `MemoryBoard` och för vaje metod som implementeras kan man göra ett enkelt test genom att anropa respektive metod från main-metoden.

3. För att visa korten och låta användaren klicka på ett kort finns den färdigskrivna klassen `MemoryWindow` (se specifikationen nedan). Klassen är en subklass till `SimpleWindow` vilket betyder att den har de attribut och metoder som finns i `MemoryWindow` plus allt som finns i `SimpleWindow`.

```
/** Skapar ett fönster som kan visa memorybrädet board. */
MemoryWindow(MemoryBoard board);

/** Ritar brädet med alla korten. */
void drawBoard();

/** Ritar kortet på rad r, kolonn c.
    Raderna och kolonnerna numreras från 0 och uppåt. */
void drawCard(int r, int c);

/** Tar reda på raden för senaste musklick. */
int getMouseRow();

/** Tar reda på kolonnen för senaste musklick. */
int getMouseCol();
```

   Vilka metoder som finns i `SimpleWindow` undersöker du enklast i ditt workspace. (I Eclipse-projektet *cs_pt* under katalogen *doc* kan man hitta filen `index.html`. Dubbelklicka på den för att få upp dokumentationen över de färdiga klasserna. Välj `SimpleWindow` i listan nere till vänster.)

   De metoder i `SimpleWindow` som du troligen behöver för att klara av uppgiften är följande:

```
/** Väntar ms millisekunder. */
static void delay(int ms);

/** Väntar tills användaren klickat på ett kort på brädet. */
void waitForMouseClick();
```

   Öppna `MemoryWindow` och läs igenom koden innan du går vidare.

4. Öppna klassen `MemoryGame` och skriv klart `main`-metoden som ska låta en person spela Memory. Korten ska visas i ett fönster och spelaren ska upprepade gånger välja två kort genom att klicka på dem. När alla kort visar framsidan är spelet slut och antal försök som krävdes ska skrivas ut.

```
public class MemoryGame {
    public static void main(String[] args) {
        String[] frontFileNames = { "can.jpg", "flopsy_mopsy_cottontail.jpg",
```

```
                "friends.jpg", "mother_ladybird.jpg", "mr_mcgregor.jpg",
                "mrs_rabbit.jpg", "mrs_tittlemouse.jpg", "radishes.jpg" };

            // Fyll i egen kod här
        }
    }
```

Vektorn `frontFileNames` innehåller de 8 filnamnen för framsidesbilderna. Bilden för kortens baksida finns i en fil med namnet *back.jpg*.

Ledning:

- Metoden `drawCard` i `MemoryWindow` ritar baksidan av det angivna kortet om det inte är vänt, och framsidan av kortet om det är noterat som vänt. Att vända ett kort är alltså i själva verket att ändra informationen om huruvida kortet är vänt eller ej, och därefter rita om kortet på nytt.
- Tänk på att spelaren måste hinna se på korten en stund innan de eventuellt vänds tillbaka.
- Om man klickar på ett redan uppvänt kort ska klicket ignoreras.

Frivilliga extrauppgifter

Extrauppgifterna har ingen inbördes ordning och beror inte av varandra.

5. Istället för att skriva ut antalet försök som krävdes till terminalfönstret kan det vara idé att visa resultatet i en ruta istället. Experimentera med Javas inbyggda klass `JOptionPane` för att göra detta. Testa t.ex:

   ```
   JOptionPane.showMessageDialog(null, "meddelande", "titel",
       JOptionPane.INFORMATION_MESSAGE);
   ```

6. Modifiera ditt spel så att användaren kan välja att köra en runda till, utan att stänga av mellan gångerna.

7. Inför ett rekord (”highscore”) som visas efter varje spelomgång och skrivs till fil.

8. För den intresserade: kör ditt Memory-program som en Android-applikation.
   Ledtrådar finns här: `http://cs.lth.se/androidmemory`