Institutionen för
Data- och informationsteknik
JSk

TENTAMEN

OBJEKTORIENTERAD PROGRAMVARUUTVECKLING

Övningstentamen 2

TID: 4 timmar

Ansvarig: Jan Skansholm

Betygsgränser: Sammanlagt maximalt 60 poäng.
På tentamen ges graderade betyg:
CTH: 3:a 24 poäng, 4:a 36 poäng, 5:a 48 poäng
GU: G 24 poäng, VG 48 poäng

Hjälpmedel: Skansholm, *Java direkt med Swing*, valfri upplaga, Studentlitteratur.
(Understrykningar och mindre anteckningar i boken är tillåtna.)

Inga kalkylatorer är tillåtna.

Tänk på:

- att skriva tydligt och disponera papperet på ett lämpligt sätt.
- att börja varje ny (del)uppgift på nytt blad. Skriv endast på en sida av papperet
- Skriv den (anonyma) kod du fått av tentamensvakten på *alla* blad.

De råd och anvisningar som givits under kursen skall följas vid programkonstruktionerna. Det innebär bl.a. att onödigt komplicerade, långa och/eller ostrukturerade lösningar i värsta fall ej bedöms.

Uppgift 1) Vart och ett av följande kodexempel innehåller ett fel, som leder till att det inte går igenom kompileringen. Hitta felet och beskriv vad det beror på, samt hur man skulle kunna lösa det på ett bra sätt. Varje klass och gränssnitt antas ligga i en egen fil med ett korrekt namn. För att underlätta är raderna numrerade. Deluppgifterna är oberoende av varandra och kan vardera ge maximalt två poäng.

(4 p)

a)
```
 1 public interface Ost {
 2    public double fetthalt(double p, double q);
 3    public void lagra();
 4 }
 5
 6 public class Greve implements Ost {
 7    public double fetthalt(double x, double y) {
 8       return (x+y) / (x*y + (1-x)*(1-y));
 9    }
10 }
```

b)
```
11 public class Fisk {
12    protected int k;
13    public Fisk(int k) {
14       this.k = k;
15    }
16 }
17
18 public class Torsk extends Fisk {
19    public Torsk(int k) {
20       this.k = k;
21    }
22 }
```

Uppgift 2) En engelsk motsvarighet till det svenska rövarspråket är *Pig Latin.* I detta språk bildas ord genom att man flyttar om och lägger till bokstäver enligt följande regler:

- Om den första bokstaven i ett ord är en vokal läggs ändelsen “yay” till sist i ordet. Ordet “are” blir t.ex. “areyay”. Observera att bokstaven y räknas som en konsonant på engelska.
- Om den första bokstaven i ett ord är en konsonant flyttas den första bokstaven till slutet av ordet och ändelsen “ay” läggs till allra sist. Ordet “can” blir t.ex. “ancay”. Undantag från denna regel är ord som börjar med vissa kombinationer av två bokstäver. Se nästa regel.
- Om den första bokstaven i ett ord är en konsonant och den andra bokstaven är någon av bokstäverna *hlprt* flyttas de två första bokstäverna till slutet av ordet och ändelsen “ay” läggs till allra sist. Ordet “think” blir t.ex. “inkthay”.

Skriv en klassmetod `toPig` som får ett engelskt ord som parameter. Som resultat skall metoden ge ordet översatt till *Pig Latin*. I resultatet skall alla eventuella stora bokstäver vara översatta till motsvarande små bokstäver. Om ordet innehåller något eller några tecken som inte är en bokstav i intervallet a-z så skall metoden generera en exception av typen `IllegalArgumentException`.

(8 p)

Uppgift 3) a) Skriv en klassmetod med namnet `ärSekvens` som kontrollerar innehållet i ett heltalsfält (array). Som parametrar skall metoden få fältet samt två heltal (`först` och `sist`) vilka anger ett heltalsintervalls första respektive sista tal. Metoden skall kontrollera att fältet existerar (inte är **`null`**), att varje heltal i intervallet finns *exakt* en gång i fältet samt att det inte finns några ytterligare element i fältet. Talen i fältet behöver dock inte ligga i storleksordning. Om fältet uppfyller kraven skall värdet **`true`** ges som resultat annars **`false`**. Vid retur från metoden måste elementen i det fält som gavs som argument ligga i sin ursprungliga ordning.

(5 p)

b) I Sudoku använder man normalt en spelplan som består av 9x9 rutor vilka kan innehålla siffrorna 1-9. Spelplanen är indelad i 9 s.k. regioner bestående av 3 rader och 3 kolumner, så som framgår av nedanstående figur.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Från början är några av rutorna ifyllda med siffror och det gäller att fylla i de övriga rutorna på så sätt att varje rad, kolumn och region kommer att innehålla siffrorna 1-9 och så varje siffra bara förekommer en gång i en viss rad, kolumn eller region.

Din uppgift är skriva en klassmetod med namnet `ärLöst` om undersöker om en Sudoku-spelplan är ifylld på ett korrekt sätt. (Du skall inte visa någon spelplan grafiskt.) Som parameter får metoden ett tvådimensionellt fält med komponenter av typen **`int`**. Du får förutsätta att fältet har 9 rader och 9 kolumner. Fältet symboliserar spelplanen och innehåller de ifyllda siffrorna. Din metod skall som resultat ge värdet **`true`** om spelplanen fyllts i på ett korrekt sätt och värdet **`false`** annars.

Tips: Använd dig av metoden `ärSekvens` från deluppgift a. Det får du göra även om du inte lös den deluppgiften.

(9 p)

Uppgift 4) Antag att klassen `Flight` är definierad på följande sätt:

```
public class Flight {
    private String no, destination;
    private int hour, min;              // avgångstid

    public Flight (String n, String d, int h, int m) {
      no = n; destination = d; hour = h; min = m;
    }

    public String getNumber() {
      return no;
    }

    public String getDestination() {
      return destination;
    }

    public int getHour() {
      return hour;
    }

    public int getMin() {
      return min;
    }
  }
```

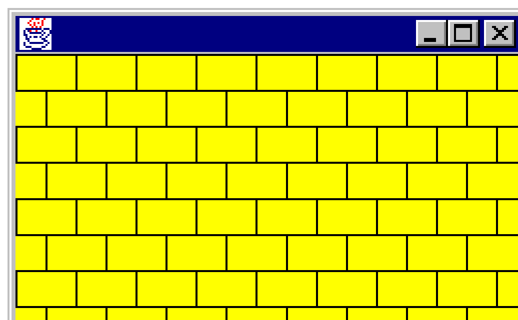
a) Komplettera klassen `Flight` så att objekt av denna klass blir naturligt jämförbara. När två objekt jämförs skall de i första hand sorteras avseende avgångstid och i andra hand avseende destinationens namn.

(5 p)

b) Konstruera en klass som gör det möjligt att använda en extern jämförare som jämför två objekt av klassen `Flight` så att de i första hand sorteras avseende destinationens namn och i andra hand avseende avgångstid.

(5 p)

Uppgift 5) Konstruera en egen grafisk komponent `Tegelvägg` som innehåller ett tegelmönster. Nedanstående figur demonstrerar hur det skall se ut. I fönstret i figuren har en enda grafisk komponent placerats: en komponent av typen `Tegelvägg`.



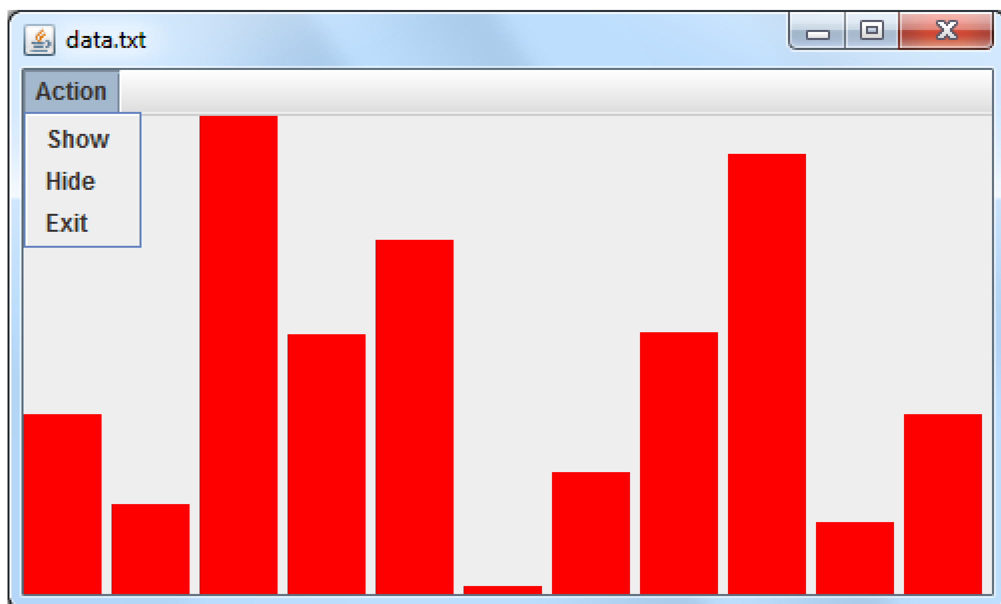
En tegelvägg skall vara gul. Tegelstenarnas bredd och höjd skall anges som parametrar till konstruktorn i klassen `Tegelvägg`. Raderna med tegelstenar skall vara förskjutna ett halvt steg i förhållande till varandra så som visas i figuren. När en `Tegelvägg`-komponent ritas skall den automatiskt känna hur stor den är och fylla hela sitt tillgängliga utrymme med tegelstenar. Du får förutsätta att komponenten inte har några ramar (`Borders`). Det behöver inte vara ett helt antal stenar på bredden eller höjden. (Det går att försöka rita utanför det tillgängliga utrymmet utan att det blir exekveringsfel och det man ritar utanför syns inte.)

(12 p)

Uppgift 6) Antag att det finns en *färdigskriven* klass `Histogram` som är en subklass till `JPanel` och som har förmågan att visa histogram på skärmen. Klassen `Histogram` har följande publika egenskaper:

- `Histogram()` skapar ett tomt histogram (utan staplar),
- `Histogram(double[] a)` skapar ett histogram med lika många staplar som antalet element i `a`. Staplarnas höjder bestäms automatiskt av elementens värden.
- `void setValues(double[] a)` tar bort diagrammets ev. tidigare staplar och lägger dit lika många staplar som antalet element i `a`. Staplarnas höjder bestäms automatiskt av elementens värden.
- `void addValue(double d)` lägger till en ny stapel sist i diagrammet. Stapelns höjd bestäms av `d`:s värde. (Eventuellt skalas då också tidigare staplar om.)

Din uppgift är att skriva ett fullständigt program som, med användning av klassen `Histogram`, kan läsa in data från en fil och presentera de data filen innehåller i form av ett histogram. Det kan t.ex. se ut som i nedanstående figur. Längst upp i fönstret skall det finnas en meny med namnet *Action*. Menyn ska ha de tre alternativen *Show*, *Hide* och *Exit*.



Om användaren väljer alternativet *Show* skall programmet först ta bort ett eventuellt tidigare diagram från fönstret. (Fönstret skall alltså bli tomt.) Därefter skall det visa en dialogruta i vilken användaren kan skriva in namnet på en textfil. Filen förväntas innehålla ett godtyckligt antal mätvärden. Det kan stå ett eller flera mätvärden på varje rad i filen. Om filen existerar skall programmet läsa indata från filen och visa ett histogram i fönstret med lika många staplar som antalet värden i filen. Dessutom skall filens namn visas i fönstrets ram. Om filen inte existerar skall programmet ge ett felmeddelande i en dialogruta.

Om användaren väljer alternativet *Hide* skall programmet se till att fönstret blir tomt och att det inte står något filnamn i fönstrets ram.

Om användaren väljer alternativet *Exit* skall programmet avslutas.

(12 p)