Institutionen för Datavetenskap
JSk

TENTAMEN

TDA540

**TENTAMEN I**

**OBJEKTORIENTERAD PROGRAMMERING för Z1**

TID 14.15 - 18.15

---

Ansvarig: Jan Skansholm

Betygsgränser: Sammanlagt maximalt 60 poäng.
På tentamen ges graderade betyg:.
3:a 24 poäng, 4:a 36 poäng och 5:a 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 uppgift på nytt blad. Skriv endast på en sida av papperet
- Skriv ditt personnummer 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) Här nedanför ser du ett felaktigt program. Avsikten är att programmet skall läsa in en text och kontrollera att den inlästa texten bara innehåller siffror. Programmet innehåller tre olika fel. Det finns ett syntaxfel som gör att det blir fel vid kompileringen. Om man rättar detta fel och provkör programmet visar det sig att det innehåller ytterligare två fel: ett logiskt fel som gör att programmet inte utför det som det skall och ett fel som kan resultera i ett exekveringsfel så att programmet avbryts. Ange vilka de tre felen är och ge för varje fel förslag på hur man kan skriva för att rätta felet!

```
// Ett avsiktligt felaktigt program
import javax.swing.*;

public class FinnTreFel {
  public static void main (String[] arg) {
    String t = JOptionPane.showInputDialog("Ett tal?");
    for (int i=1; i<=t.length(); i++)
      if (t.charAt(i) >= '0' &&  <= '9')
        JOptionPane.showMessageDialog(null,"Talet är OK");
      else
        JOptionPane.showMessageDialog(null,"Inget tal");
    System.exit(0);
  }
}
```

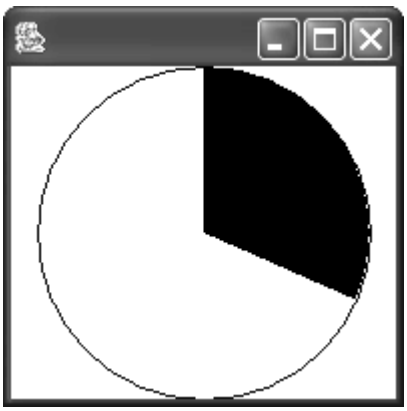
(6 p)

Uppgift 2) Ett primtal är ett tal som endast är jämnt delbart med 1 och med sig själv. En teknik att finna primtal går under namnet *Eratosthenes såll*. Den går till på följande sätt: Man använder ett fält där elementen har typen **`boolean`**. Från början sätts element nummer 0 och 1 till **`false`** och alla övriga element i fältet till **`true`**. Därefter löper man igenom fältet med början på index 2. När man stöter på ett element (låt oss säga att det har nummer *i*) som har värdet **`true`** gör man följande. Löp igenom resten av fältet (fr.o.m. *i*+1 till fältets slut) och sätt varje element som har den egenskapen att dess index är en jämn multipel av *i* till **`false`**. När man t.ex. kommer till element nr. 5 så skall elementen 10, 15, 20 etc. i fältet ges värdet **`false`**.

När denna process är avslutad visar de element i fältet som fortfarande har värdet **`true`** vilka index som är primtal. Konstruera ett program som använder denna teknik för att beräkna och skriva ut alla primtal som är mindre än eller lika med 1000. Utskriften skall ske i ett kommandofönster.

(10 p)

Uppgift 3) Uppgiften är att konstruera en egen grafisk klass `MyTimer`. Ett objekt av denna klass skall fungera som en äggklocka. Det skall också kunna fungera som en Swing-komponent. I figuren nedan ser du hur det kan se ut när man har placerat ut ett `MyTimer`-objekt som enda komponent i ett fönster (Observera att du inte skall skriva ett program som visar fönstret. Du skall bara konstruera klassen `MyTimer`.).



När man startar ett `MyTimer`-objekt sätts en del av (eller hela) cirkeln till svart. Därefter håller `MyTimer`-objektet själv reda på tiden och låter den svarta delen bli allt mindre tills den slutligen helt försvinner. (Det ser ut som en visare som går motsols.) När detta sker skall `MyTimer`-objektet ge alarm. Det gör det genom att dels ge ifrån sig ett pling-ljud och dels fylla hela cirkeln med röd färg.

Tips: Du får gärna utgå från klassen `CircleDiagram` på sidan 250 i kursboken (sidan 238 i äldre upplagor). Om du använder denna klass som den är behöver du inte skriva om den.

Det skall finnas två konstruktorer i klassen `MyTimer`. Den första har ett heltal som parameter, vilket anger hur lång tid ett helt varv i cirkeln motsvarar. Detta anges i sekunder. Parametern 60 anger t.ex. att ett helt varv motsvarar en minut. Den andra konstruktorn skall vara parameterlös. Den skall initiera det nya objektet så att ett helt varv motsvarar en timme.

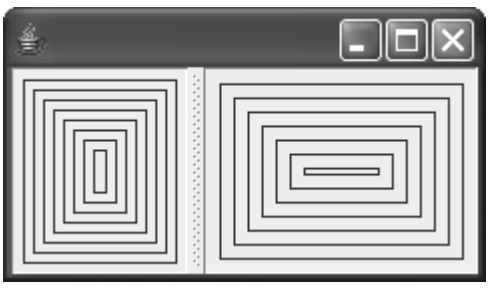
Följande metoder skall kunna anropas utifrån:

- `start(t)`. Startar `MyTimer`-objektet så att det kommer att ge alarm om `t` sekunder. Fyller i så stor del av cirkeln som tiden `t` motsvarar. (Om `t` t.ex. är 45 sekunder och ett helt varv motsvarar en minut så skall tre fjärdedelar av cirkeln fyllas i.) Cirkeln skall fyllas i med svart färg. Metoden `start` skall vara så utformad att det går att anropa den även om `MyTimer`-objektet redan skulle vara igång eller vara stoppat. Då skall det återstartas.
- `clear()`. Stoppar `MyTimer`-objektet så att det inte kommer ett ge alarm. Ser också till att hela cirkeln blir tom och att cirkelns omkrets ritas med svart färg.

Utforma klassen `MyTimer` så att cirkeln ritas om en gång per sekund. Varje gång den ritas om skall den svarta delen blir mindre och mindre. När tiden gått ut skall man ge alarm på det sätt som beskrevs ovan.

(11 p)

Uppgift 4) Nedanstående figur visar ett fönster som (med hjälp av en `JSplitPane`) delats upp i två delar. Varje del innehåller en grafisk komponent av klassen `BoxPanel`. Din uppgift är att konstruera klassen `BoxPanel`. Observera att du *inte* behöver konstruera hela programmet som genererar fönstret.



Klassen `BoxPanel` skall vara en subklass till standardklassen `JPanel`. En komponent av klassen `BoxPanel` skall automatiskt rita ett antal rektanglar som ligger inne i varandra så som demonstreras i figuren. Rektanglarna skall fylla ut hela komponentens utrymme. Avståndet (i pixlar) mellan två linjer skall ges som en parameter till klassens konstruktor. (I den vänstra komponenten i figuren är t.ex. avståndet 5 pixlar och i den högra 7.)

Ritning av rektanglarna skall förstås ske när metoden `paintComponent` anropas. Detta *skall* göras i en *rekursiv* metod `drawSquares` vilken anropas från metoden `paintComponent`. Metoden `drawSquares` har följande specifikation:

```
public void drawSquares(Graphics g, int x, int y, int w, int h)
```

Parametrarna `x` och `y` anger koordinaterna för nästa rektangels övre vänstra hörn och parametrarna `w` och `h` anger det återstående utrymmets bredd resp. höjd. Varje rekursivt anrop skall rita en rektangel. Ritningarna skall pågå så länge det finns återstående utrymme att rita på.

Din klass `BoxPanel` skall alltså innehålla en konstruktor samt metoderna `paintComponent` och `drawSquares`.

(10 p)

Uppgift 5) Antag att klassen `Flight` är definierad som på sidan 135 (117) i kursboken och att klassen `Tidpunkt` är definierad som på sidorna 68 (64) och 84 (80). (Sidorna inom parentes gäller för upplagorna 3 och 4 av boken.) Antag vidare att klassen `Tidpunkt` är utökad så att objekt av denna klass är naturligt jämförbara och att jämförelserna sker i tidsordning. När du löser en viss deluppgift i denna uppgift får du förutsätta att de föregående deluppgifterna är lösta, även om du inte löst dem själv.

a) Utöka klassen `Flight` så att objekt av denna typ blir naturligt jämförbara. Jämförelserna skall ske i tidsordning avseende avgångstiden. Om två flighter har samma avgångstid skall de hamna i bokstavsordning avseende namnet på destinationen. (För enkelhets skull bortser vi från att det även finns ett flightnummer.) Du behöver inte skriva om hela klassen `Flight`. Visa bara tydligt vilka tillägg som behövs.

(4 p)

b) Gör ytterligare ett tillägg i klassen `Flight`. Lägg till en metod `equals` som omdefinierar den standardversion av `equals` som ärvs från klassen `Object`. Två flighter skall anses vara lika om de har samma destination och avgångstid. (Bortse även här från att det finns ett flightnummer.) Tänk på att metoden `equals` som parameter får en referens till ett objekt av typen `Object`. Om detta objekt inte är av klassen `Flight` så är det förstås inte lika med det aktuella objektet.

(3 p)

c) Konstruera en klass `Airport`. För varje objekt av denna klass skall det finnas tre instansvariabler: flyplatsens namn, en mängd `departures` med alla flighter som avgår från den aktuella flygplatsen samt en avbildningstabell `flightsTo`. Mängden `departures` skall vara sorterad i första hand på avgångstid och i andra hand på destinationens namn. I tabellen `flightsTo` skall söknycklarna vara namnen på destinationerna och värdena mängder innehållande flighter. Man skall alltså i tabellen `flightsTo` kunna slå upp vilka flighter som avgår till en viss destination. Varje värdemängd i `flightsTo` skall vara sorterad på avgångstid.

En konsekvens av att det finns både en mängd med alla avgångar och en tabell med avgångar till varje destination är att viss fligth kommer att ingå i två mängder. Men detta är inget konstigt eftersom det i själva verket är referenserna till flighterna som ligger i mängderna.

Klassen `Airport` skall ha en konstruktor som får flygplatsens namn som parameter. Dessutom skall följande metoder finnas:

- `getName`, ger flyplatsens namn,
- `getDepartures()`, ger en sorterad mängd innehållande alla flighter som avgår från en aktuella flygplatsen,
- `getDepartures(dest)`, ger en sorterad mängd innehållande alla flighter som avgår till `dest` från en aktuella flygplatsen (`dest` är en `String`),
- `addFlight(f)`, lägger till flighten `f` både till mängden `departures` och till aktuell mängd i tabellen `flightsTo`,
- `removeFlight(f)`, tar bort flighten `f` både från mängden `departures` och från aktuell mängd i tabellen `flightsTo`. Om detta innebär att en viss avbildning i tabellen `flightsTo` kommer att innehålla en tom mängd så skall denna avbildning tas bort.

När du konstruerar metoderna `getDepartures` behöver du för enkelhets skull inte ta hänsyn till att det kan vara farligt att returnera referenser.

(8 p)

d) Antag att det finns en textfil `flights.txt` som innehåller information om flighter. Varje rad i filen innehåller information om en flight (flightnummer, från, till, avgångstid och ankomsttid). En rad kan t.ex. se ut på följande sätt:

```
SK123 Göteborg Köpenhamn 8 10 8 50
```

Det kan finnas ett godtyckligt antal rader filen och raderna är inte sorterade på något sätt. Uppgiften är att skriva ett program som läser in informationen i filen och lägger den i en avbildningstabell där nycklarna är avgångsflygplatsens namn och värdena är objekt av typen `Airport`. (Varje gång man påträffar en ny avgångsflygplats som inte funnits med tidigare skall man alltså skapa en ny ingång i tabellen.)

När all information har lästs in och lagts i avbildningstabellen skall denna lagras, som ett enda objekt, i en fil med namnet `flights.dat`. (Tips: Använd klassen `ObjectOutputStream`.) I denna deluppgift får du förutsätta att klasserna `Tidpunkt`, `Fligth` och `Airport` har kompletterats så att de även implementerar gränssnittet `Serializable`.

(8 p)