Data och Informationsteknik / Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
Robin Adams

Göteborg 8 June 2018

PROGRAMMERINGSTEKNIK TIN212

Dag: Fredag Datum: 2018-06-08 Tid: 8.30-13.30 (OBS 5 tim)

Ansvarig lärare: Robin Adams tel. 0768-564864

Tentamen är uppdelad i två delar. Om du har klarat Duggan behöver du lösa bara Del 2. Om du har inte klarat Duggan så måste du lösa både Del 1 och del2.

Betygsgränser:

Om du har klarat Duggan: 3 = 15p, 4 = 20p, 5 = 25p, max 30p

Om du har inte klarat Duggan: 3 = 28p, 4 = 38p, 5 = 48p, max 60p

Hjälpmedel:

Skansholm, *Java direkt med Swing,* Studentlitteratur, eller
Bravaco, Simonson, *Java Programming: From the Ground up*
(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 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.

DEL 1

Svara på dessa uppgifter endast om du **inte** har klarat Duggan.

Uppgift 1 (10 poäng)

a) (3p) Vad blir utskriften från nedanstående program? (Vad innehåller arr I slutet av main-metoden?)

```
public class Uppgift1a{
   public static void mystery (int[] a, int n) {
      for (int i = 0; i < a.length*2; i++) {
         a[i%a.length] = a[i%a.length] + a[(i+1)%a.length];
      }
   }

   public static void main (String[] args) {
      int[] arr = {1, 2, 3};
      mystery(arr, 2);
      System.out.println(java.util.Arrays.toString(arr));
   }
}
```

b) (3p) Vad blir utskriften från nedanstående program? (Notera: tre utskrifter!)

```
public class Uppgift1b {
   public static boolean gb = false;
   public boolean b = false;

   public void f(boolean q){
      b = b || q;
      gb = !gb;
   }

   public static void main(String[] args) {
      new Uppgift2b();
      Uppgift2b x = new Uppgift2b();
      Uppgift2b y = new Uppgift2b();
      System.out.println(x.b || y.b || gb);
      new Uppgift2b();
      x.f(true);
      y.f(x.b);
      System.out.println(y.b);
      System.out.println(gb);
   }
}
```

c) (2p) Betrakta nedanstående metod:

```
public static void reverse(int[] arr){
   for (int i = 0; i < arr.length; i++){
      int temp = arr[i];
      arr[i] = arr[arr.length-i-1];
      arr[arr.length-i-1] = temp;
   }
}
```

Programmeraren hade tänkt att den skulle vända ordningen på värdena i en array, men programmet fungerar inte alls och resultatet av en testkörning visade sig högst förvirrande. Svara på nedanstående:

- Beskriv kortfattat (1-2 meningar) vad metoden gör i dess nuvarande form
- Hur kan man åtgärda problemet (vad som ska ändras)?

d) (2p) Betrakta följande enkla Java-kod:

```
public class Uppgift1d {
   public void printDirectory(){
      System.out.println("Directory is: c:\new\");
   }
}
```

När programmet kompileras uppstår inte mindre än tre kompileringsfel:

```
Uppgift2d.java:3: error: unclosed string literal
      System.out.println("Directory is: c:\new\");
                         ^
Uppgift2d.java:3: error: ';' expected
      System.out.println("Directory is: c:\new\");
                                                 ^
Uppgift2d.java:5: error: reached end of file while parsing
}
 ^
```

Alla fel beror på ett enda misstag, förklara kortfattat vad det är och hur man rättar till det.

Uppgift 2 (10 poäng)

Ett primtal är ett tal större än 1 som saknar delare andra än 1 och talet självt. Till exempel 2 och 5 är primtal men 6 är inte ett primtal eftersom det är delbart med 2 och 3. De första 10 primtalen är: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29.

a) (5 p) Skriv en metod `public static boolean isPrime(int x)` om x är ett primtal (returnerar **`true`** om `x` är ett primtal, annars **`false`**). Ni testar det genom att dela med alla tal, men för att snabba upp funktionen observerar vi följande:

   1. Du behöver inte testa jämna tal förutom 2 (om ett tal inte är delbart med 2 är det inte delbart med något jämnt tal).
   2. Ni behöver inte testa något tal större än kvadratroten ur `x` (om *x=y*z* måste minst en av *y* och *z* vara $\leq \sqrt{x}$)

   Detta ger oss följande ungefärliga algoritm för funktionen:

   - Testa om `x` är mindre än eller lika med 1 (i så fall är `x` inte ett primtal)
   - Testa om `x` är delbart med 2 (i så fall är `x` inte ett primtal, förutom ifall `x==2`).
   - Testa sedan om `x` är delbart med något ojämnt heltal från 3 upp till och med $\sqrt{x}$ (avrundat nedåt).
   - I annat fall är `x` ett primtal.

   Undvik att räkna ut kvadratroten av `x` mer än en gång (det skulle sakta ner programmet i onödan).

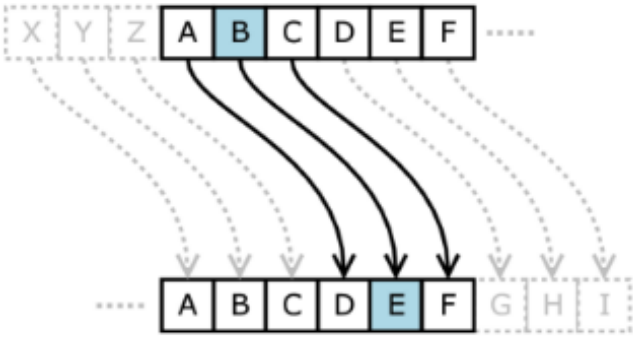
b) (5 p) Skriv en `main`-metod som läser in ett tal större än 1 från användaren (`System.in` eller en dialogruta), och skriver ut alla *primtalsfaktorer* för det talet i fallande ordning (från högsta till lägsta). Utskriften sker lämpligen till `System.out`, men en dialogruta är också OK. Till exempel för 198 ska den skriva ut talen 11 3 3 2 eftersom 11*3*3*2=198. Alla talen som skrivs ut ska vara primtal, och deras produkt ska bli det inmatade talet (det finns bara en lösning för varje tal!). Tänk på at för till exempel 20 ska programmet skriva ut 5 2 2, inte 5 4, eftersom 4 inte är ett primtal. För inmatningen 13 ska programmet skriva ut 13 eftersom det är den enda primtalsfaktorn. Du behöver inte hantera några felaktiga inmatningar.

   Tips:

   1. För talet 198 är t=11 det högsta talet där `198%t==0`, och t är ett primtal.
   2. Primtalsfaktorerna för 198/11=18 är 3 3 2, primtalsfaktorerna för 18/3=6 är 3 2 och så vidare.
   3. Du får använda `isPrime` från uppgift a).

Uppgift 3 (10 poang)

Ett Caesarshiffer är ett mycket enkelt sätt att kryptera meddelanden som går ut på att man har en heltalsnyckel *k*, och helt enkelt byter ut varje bokstav mot den bokstav som ligger *k* steg länge fram i alfabetet. Med exempelvis nyckeln *k*=3, blir texten ”abc” ---> ”def”. Om vi kommer till slutet av alfabetet börjar vi helt enkelt om från början, så med *k*=3 blir ”ääö” ---> ”abc” (bilden nedan visar det engelska alfabetet som saknar ÅÄÖ, men principen är densamma).



Denna uppgift går ut på att skriva ett program `Uppgift9.java` som läser in en textfil, krypterar den med ett Caesarshiffer och skriver den krypterade texten till en annan fil. För enkelhets skull antar vi att textfilen enbart innehåller ord innehållande bokstäver a-ö separerade med blanksteg (inga siffror, skiljetecken eller andra symboler).

Du kan utgå från att alfabetet (strängen `ALPHABET` nedan) som programmet kan förväntas hantera redan är definierad i klassen `Uppgift9` (du behöver alltså inte skriva den själv):

```
public static final String ALPHABET = "abcdefghijklmnopqrstuvwxyzääö";
```

Programmet ska inte skilja på stora och små bokstäver (d.v.s. ’A’ och ’a’ ska ses som samma bokstav) och den krypterade texten behöver bara innehålla små bokstäver. När man startar programmet ska man på kommandoraden ange tre argument: filnamnet för indatan, filnamnet för utdatan samt en (positiv) heltalsnyckel att använda i Caesarshiffret:

```
java Uppgift9.java Uppg9In.txt Uppg9Ut.txt 1
```

Om filen `Uppg9In.txt` innehåller texten:
`Apa Katt Hund Örn`
ska filen `Uppg9Ut.txt` efter kommandot ovan innehålla den krypterade texten:
`bqb lbuu ivoe aso`

Programmet ska kontrollera att antalet argument är korrekt, samt att filerna går att öppna. Om något skulle vara fel ska en felutskrift ges och programmet avslutas.

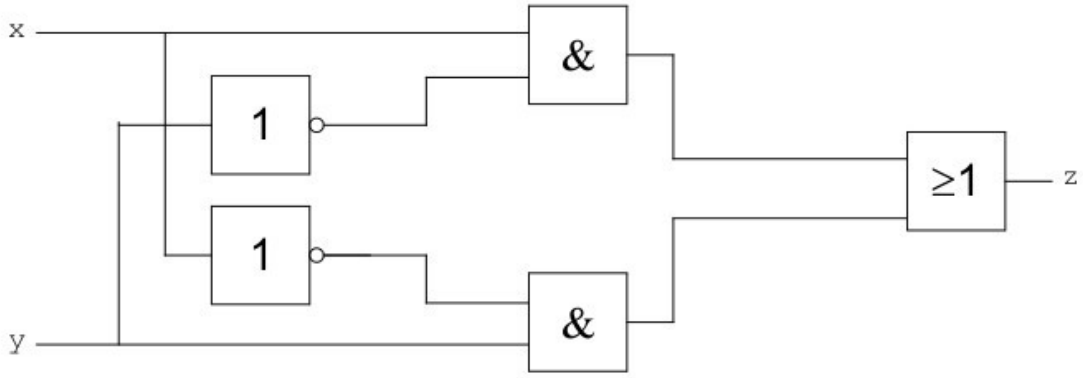
Tips: Skriv en separat hjälpmetod `String encrypt(String plainTxt, int key)` som krypterar en `String` med en given nyckel först. Använd sen denna i huvudprogrammet.

DEL 2

Alle studenter måste svara på denna.

Uppgift 4 (10 poang)

När man konstruerar digitala system utgår man som bekant från enkla elektroniska kretsar som brukar kallas grindar (gates). Grindarnas in- och utsignaler är elektriska spänningar på "låg nivå" eller "hög nivå" (false resp. true). De enklaste grindarna avbildar på elektronisk väg de logiska funktionerna "icke", "och" och "eller" och kallas därför NOT-grindar, AND-grindar resp. OR-grindar. Ett exempel på ett nät som använder sådana grindar visas i figuren nedan. (Grindarna markerade med symbolen 1 är NOT-grindar, de med symbolen & AND-grindar och den med beteckningen ≥1 en OR-grind.)



Vi vill också att kretsdesignern ska kunna ange att (t. ex.) signalen vid x alltid ska vara true. Så vi tillåter designern att inkludera konstanta insignaler till en krets. Vi tänker på en konstant insignal som en speciell typ av grind med 0 ingångar. En konstant insignal "true" är en grind med 0 ingångar och vars utgång är alltid true.

Konstruera en grupp klasser som beskriver de grundläggande grindarna samt konstanta insignaler. Du skall skapa en abstrakt superklass `Gate` och utgå från denna. Klassen `Gate` skall ha en abstrakt subklass `BasicGate` vilken i sin tur skall vara superklass till de tre klasserna `NotGate`, `AndGate` och `OrGate`. Dessutom skall du konstruera en klass `ConstantGate` som beskriver konstanta insignaler till nätet. Denna klass skall vara direkt subklass till klassen `Gate`.

När man ritar upp komplicerade nät som innehåller grindar måste man kunna namnge signalerna och grindarna. Därför skall klassen `Gate` ha en instansvariabel som innehåller grindens (eller insignalens) namn. Dessutom skall alla de klasser du konstruerar ha en konstruktor som har ett namn som parameter.

Superklassen `Gate` skall ha följande metoder:

- `getName`, ger namnet som resultat,
- `getOutput`, abstrakt metod som skall ge utsignalen (en `boolean`) från den aktuella grinden (eller insignalens värde),

Klassen `ConstantGate` används för att beskriva konstanta insignaler till de nät man vill koppla upp. (Man kan tänka sig en insignal som en "låtsasgrind" som alltid genererar en

konstant utsignal.) I klassen `ConstantGate` måste man förstås omdefiniera metoden `getOutput`. Dessutom skall klassen `ConstantGate` ha en metod som heter `set`. Denna skall ha en parameter av typen `boolean` som anger den konstanta insignalens värde.

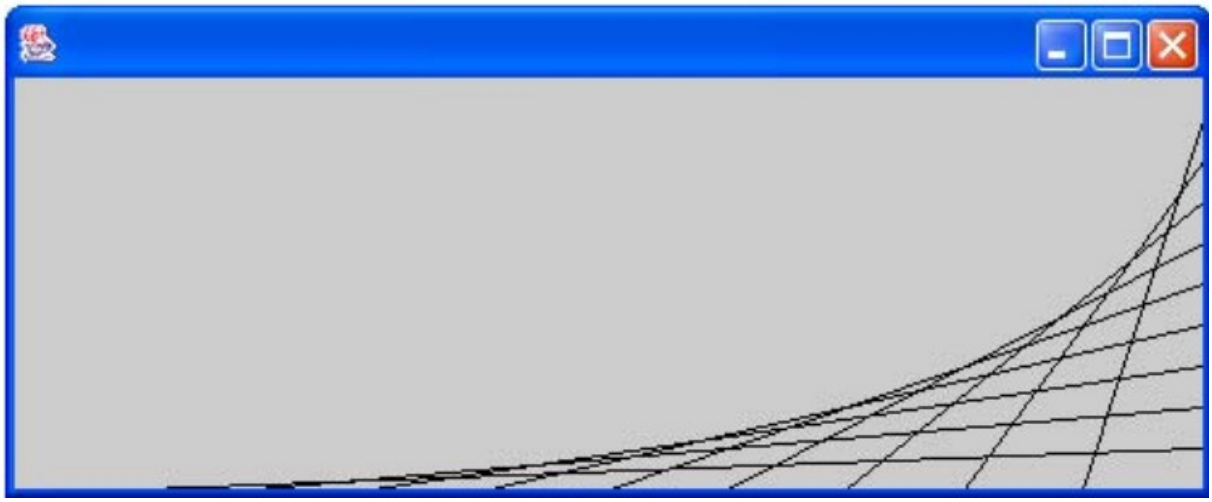
Varje objekt av klassen `BasicGate` skall innehålla en lista med referenser till de grindar som är kopplade till ingångarna till den aktuella grinden. Klassen `BasicGate` skall ha följande metoder av vilka vissa kan vara abstrakta:

- `setInput`, får som parameter en referens `g` av typen `Gate`, kopplar utsignalen från `g` som insignal till den aktuella grinden,
- `checkInput`, kontrollerar att den aktuella grinden har korrekt antal insignaler. (AND- och OR-grindar måste ha minst två insignaler och NOT-grindar måste ha exakt en insignal.) Om insignalerna är felaktiga skall en exception av typen `IllegalStateException` genereras. Lägg in ett lämpligt felmeddelande.
- `CalculateValue` förutsätter att antalet insignaler är korrekt och beräknar och returnerar utsignalens värde utgående från insignalerna,
- `getOutput`, ger utsignalen från den aktuella grinden (anropar först `checkInput` och sedan `calculateValue`).

I klasserna `AndGate`, `OrGate` och `NotGate` skall du överskugga den eller de ärvda metoder som behövs för att grindarna skall ge rätt resultat.

Uppgift 5 (10 poang)

Fönstret i figuren nedan innehåller en enda grafisk komponent vilken fyller ut hela fönstret. Denna komponent är av klassen `Lines`. Klassen `Lines` är en subklass till standardklassen `JPanel`. Klassen `Lines` ritar linjer så som visas i figuren. Avstånden mellan linjernas startpunkter är lika stora. Detta gäller också för linjerna slutpunkter. Antalet linjer som skall ritas anger man som parameter till konstruktorn för klassen `Lines`. (I figuren är det 9 linjer men detta kan alltså variera.) Om storlekenpå en komponent av typen `Lines` ändras (fönstrets storlek kan t.ex. ändras i figuren) så ändrar sig linjerna så att de ändå fyller ut hela komponenten. Din uppgift är nu naturligtvis att skriva klassen `Lines`. (Du behöver alltså inte skriva hela programmet). Tänk på att det kan finnas en ram (`Border`) runt en `Lines`-komponent. Det måste du förstås ta hänsyn till när du beräknar start- och ändpunkterna för linjerna.



Uppgift 6 (10 poang)

Antag att klassen `Flight` är definierad på följande sätt:

```
import java.text.*;
import java.io.*;

public class Flight implements Serializable, Comparable<Flight> {
  private String no, destination;
  private String time;                  // avgångstid (med formen hh:mm)

  public Flight (String n, String d, String t) {
    no = n;
    destination = d;
    time = t;
  }

  public String getNumber() {
    return no;
  }

  public String getDestination() {
    return destination;
  }

  public String getTime() {
    return time;
  }

  public int compareTo(Flight f) {
    Collator c = Collator.getInstance();
    c.setStrength(Collator.PRIMARY);
    return c.compare(time+destination, f.time+f.destination);
  }
}
```

Konstruera en klass `Airport`. För varje objekt av klassen skall det finnas tre instansvariabler: flygplatsens 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.

Eftersom det finns både en mängd med alla avgångar och en tabell med avgångar till varje destination kommer en viss flight att ingå i två mängder. Men detta är inget konstigt eftersom det 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.