```
// Uppgift 1

Kompileringsfel: Uttrycket (t.charAt(i) >= '0' &&  <= '9') är felaktigt,
                 skall skrivas (t.charAt(i) >= '0' &&  t.charAt(i) <= '9')
Logiskt fel:
  Programmet visar en dialogruta för varje korrekt tecken samt för det första
felaktiga.
  Förslag till rättelse:

  public static void main (String[] arg) {
    String t = JOptionPane.showInputDialog("Ett tal?");
    boolean ok = true;    // korrekt, än så länge
    for (int i=1; i<=t.length(); i++)
      if (t.charAt(i) < ’0’ || t.charAt(i) > ’9’) {
        ok = false;       // inte längre korrekt
        break;
      }
    if (ok)
      JOptionPane.showMessageDialog(null,"Talet är OK");
    else
      JOptionPane.showMessageDialog(null,"Inget tal");
    System.exit(0);
  }

Exekveringsfel: Indexering sker från 0, vilket betyder att uttrycken i
for-satsen är fleaktiga
                Skall  vara:
                for (int i=0; i<t.length(); i++)


// Uppgift 2

public class Era {
  public static void main(String[] arg) {
    final int  max = 1001;
    boolean[] a = new boolean[max];
    a[0] = a[1] = false;
    for (int k=2; k<max; k++)
      a[k] = true;
    for (int i=2; i<max; i++)
      if (a[i])
        for (int j=i+1; j<max; j++)
          if (j % i == 0)
            a[j] = false;
    for (int l=0; l<max; l++)
      if (a[l])
        System.out.println(l);
  }
}


// Uppgift 3

 import javax.swing.*;
 import java.awt.*;
 import java.awt.event.*;

 public class MyTimer extends CircleDiagram implements ActionListener {

   private javax.swing.Timer t = new javax.swing.Timer(1000, this);
   private int max;

   public MyTimer(int maxtime) {
     super(0, maxtime);
     max = maxtime;
   }
```

```
   public MyTimer() {
     this(3600);
   }

   public void clear() {
     t.stop();
     setValue(0);
     setForeground(Color.black);
   }

   public void start(int time) {
     clear();
     setValue(time);
     t.restart();
   }

   public void actionPerformed(ActionEvent e) {
     setValue(getValue()-1);
     if (getValue() == 0) {
       t.stop();
       setForeground(Color.red);
       setValue(max);
       Toolkit.getDefaultToolkit().beep();
     }
   }
}



// Uppgift 4

public class BoxPanel extends JPanel {
    private int delta;

    public BoxPanel(int d) {
        delta = d;
    }

    public void drawSquares(Graphics g, int x, int y, int w, int h) {
        if (w > 0 && h > 0) {
            g.drawRect(x, y, w, h);
            drawSquares(g, x+delta, y+delta, w-2*delta, h-2*delta);
        }
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        drawSquares(g, delta, delta, getWidth()-2*delta, getHeight()-2*delta);
    }
}


// Uppgift 5

// a
 import java.text.*;

 public class Flight implements Comparable<Flight> {
    ...

    public int compareTo(Flight f) {
     int i = dep.compareTo(f.dep);
     if (i != 0)
       return i;
     else {
       Collator c = Collator.getInstance();
       c.setStrength(Collator.PRIMARY);
```

```
      return c.compare(destination, f.destination);
    }
  }
 }

// b

   public boolean equals(Object obj) {
     if (obj instanceof Flight) {
       Flight f = (Flight) obj;
       return this.compareTo(f) == 0;
     }
     else
       return false;
   }

// c

import java.util.*;

public class Airport {
  private String name;
  private SortedSet<Flight> departures = new TreeSet<Flight>();
  private Map<String, SortedSet<Flight>> flightsTo = new HashMap<String,
SortedSet<Flight>>();

  public Airport(String n) {
    name = n;
  }

  public String getName() {
    return name;
  }

  public SortedSet<Flight> getDepartures() {
    return departures;
  }

  public SortedSet<Flight> getDepartures(String to) {
    return flightsTo.get(to);
  }

  public void addFlight(Flight f) {
    departures.add(f);
    if (!flightsTo.containsKey(f.getDestination()))
     flightsTo.put(f.getDestination(), new TreeSet<Flight>());
    flightsTo.get(f.getDestination()).add(f);
  }

  public void removeFlight(Flight f) {
    departures.remove(f);
     SortedSet<Flight> e = flightsTo.get(f.getDestination());
     if (e != null) {
       e.remove(f);
       if (e.isEmpty())
         flightsTo.remove(f.getDestination());
     }
  }
}

// d

import javax.swing.*;
import java.io.*;
import java.util.*;
import static javax.swing.JOptionPane.*;

public class CreateFlights {
```

```
  public static void main(String arg[]) throws IOException {
    ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream("flights.dat"));
    Map<String, Airport> a = new HashMap<String, Airport>();
    Scanner sc = new Scanner(new File("flights.txt"));
    while (sc.hasNext()) {
      String no = sc.next();
      String from = sc.next();
      String dest = sc.next();
      int depH = sc.nextInt(), depM = sc.nextInt(),
          arrH = sc.nextInt(), arrM = sc. nextInt();
      if (!a.containsKey(from))
        a.put(from, new Airport(from));
      a.get(from).addFlight(new Flight(no, dest, depH, depM, arrH, arrM));
    }
    out.writeObject(a);
  }
}
```