```
 1
 2  /*
 3   * @(#)Pgm1.java        1.0 2013-01-30
 4   *
 5   * Funktionalitet
 6   * Betala rÃ¤kningar om fÃ¶rfallodatum nÃ¥tts (betaldatum Ã¤r dagens datum) och om
 7   * det finns tillrÃ¤ckligt med pengar pÃ¥ kontot. NÃ¤r en betalning gjorts sÃ¥
 8   * skall betalningen tas bort frÃ¥n bevakning och endast kunna hittas i
 9   * transaktionsloggen.
10   *
11   */
12
13  import java.io.*;
14  import java.util.*;
15
16  /**
17   * I denna klass skall...
18   * @version      1.0 30 Jan 2013
19   * @author       Jonathan, Oskar, Magnus
20   */
21  public class Pgm1 {
22      public static void main(String[] args) {
23              Metoder m;
24          System.out.println("\n-= Pgm 1, Betalar rakningar =-");
25          System.out.println("Utfor transaktioner i bevakningsfilen mindre");
26          System.out.println("an en vecka gamla om det finns tackning.\n");
27
28          // Ladda datafilerna
29          try {
30              m = Metoder.buildMetoder();
31          } catch (IOException e) {
32              System.out.println("Fel vid läsning av någon av de angivna"
33                                 + " datafilerna, var god försök igen. \n"
34                                 + "Sökväg: " + new File("").getAbsolutePath() + "\n"
35                                 + "Undantag: " + e.getMessage());
36              return;
37          }
38
39          // Skapa en Date en vecka tillbaka i tiden
40          Calendar c = Calendar.getInstance();
41          c.add(Calendar.WEEK_OF_YEAR, -1);
42          Date lastWeek = c.getTime();
43
44          // Utför alla transaktioner en vecka tillbaka fram till idag
45          try {
46              m.executeAllTransactionsBetween(lastWeek, new Date());
47          } catch (Exception e) {
48              System.out.println("Kunde inte utföra transaktionerna:\n" + e.getMessage());
49          }
50
51          // Sparar alla ändringar
52          try {
53              m.saveChanges();
54          } catch (IOException e) {
55              System.out.println("Kunde inte spara Ã¤ndringar:\n" + e.getMessage());
56          }
57          // Programmet slutar
58          System.out.println("Finished.");
59      }
60  }
61
```

```
 62  /**
 63   *
 64   * Maintains accounts. Here we can list our accounts and any info connected to
 65   * them, make new accounts or create transactions.
 66   *
 67   * @author Jonathan SkÃ¥rstedt
 68   * @author Oskar PÃ¥lsgÃ¥rd
 69   * @author Magnus duberg
 70   *
 71   * @version 1.0
 72   */
 73
 74  import java.io.*;
 75  import java.text.*;
 76  import java.util.*;
 77
 78
 79  public class Pgm2 {
 80
 81              private static Scanner tbScanner = new Scanner(System.in);
 82              private static Metoder m;
 83
 84              public static void main(String[] args){
 85          System.out.println("\n-= Pgm2, Konto- och transaktionshantering =-\n");
 86
 87                          try {
 88                                      m = Metoder.buildMetoder();
 89                          } catch (IOException e) {
 90              System.out.println("Kunde inte hitta en eller flera av de angivna"
 91                          + " filerna, var god forsok igen. \n"
 92                          + "Aktuell mapp: " + new File("").getAbsolutePath() + "\n"
 93                          + e.getMessage());
 94              return;
 95                          }
 96
 97                          String huvudMeny =
 98                                      "=======================================\n" +
 99                              "== Meny ================================\n" +
100                              "=======================================\n" +
101                                      "1. Lista konto        4. Ta ut pengar\n" +
102                                      "2. Skapa nytt konto   5. Registrera ny transaktion\n"
103  +
104                                      "3. Satt in pengar     0. Avsluta\n" +
105                                      "Gor ditt val: ";
106
107                          do {
108                                      System.out.print(huvudMeny);
109                                      switch (tbScanner.nextLine()){
110                                      case "1": listaKonton(); break;
111                                      case "2": skapaKonto(); break;
112                                      case "3": sattInPengar(); break;
113                                      case "4": taUtPengar(); break;
114                                      case "5": registreraTransaktion();break;
115                                      case "0":
116                                                  try {m.saveChanges();
117                                                  } catch (IOException e) {
118                                                              System.out.println("Kunde inte
119  spara till fil!\n" + e.getMessage());
120                                                  }
121                                                  System.out.println("Avslutar.");
122                                                  System.exit(0);
123                                      default:
```

```
                                        System.out.println("Forsok igen! (0-5)");
                                        break;}
                        }while(true);
        }

        /**
         * Shows information of a given account or, if no account is given, all
         * accounts.
         */
        private static void listaKonton() {
                        System.out.println("Ange kontonummer for detaljuppgifter eller
                        tryck enter for kontolistning: ");
                        String accountNumber = tbScanner.nextLine();

                        if (accountNumber.trim().length() > 0) {
                                        try {
                                                        Konto k = m.findAccount(accountNumber);


        System.out.println(Metoder.accountToString(k));
                                        } catch (NoSuchFieldException e){
                                                        System.out.println("Kontot hittades
                                                        inte!");
                                        }

                        } else {
                                        System.out.println("Visar alla kontonummer: ");
                                        Konto[] accounts = m.getAccounts();
                                        for(Konto k : accounts) {
                                                        if(k == null)
                                                                        break;


        System.out.println(Metoder.accountToString(k));
                                        }
                        }
        }

        /**
         * Add new account to our bank.
         */
        private static void skapaKonto() {
                        String number, name, owner;
                        double amount;
                        String val;
                        Konto k;
                        Random Numb = new Random();

                        System.out.print("Ange kontonummer eller tryck enter " +
                                                        "for ett slumpat kontonummer :");
                        do {
                                        val = tbScanner.nextLine();
                                        if (val.length() > 0) {
                                                        number = val;
                                                        System.out.println("sparar " + number);

                                        } else { // Slumpar fram ett kontonummer
                                                        int x = Numb.nextInt(10000);
                                                        int y = Numb.nextInt(10000000);
                                                        number = x + "-" + y;
                                                        System.out.println("slumpar.. " + number);
                                        }
```

```
                                    if(m.accountExists(number))
                                                System.out.print("Konto existerar. ");
                        } while(m.accountExists(number));

                        System.out.print("Skriv in kontonamn: ");
                        name = tbScanner.nextLine();

                        System.out.print("Skriv in Agarens namn: ");
                        owner = tbScanner.nextLine();

                        System.out.print("Skriv in saldo: ");
                        amount = Double.parseDouble(tbScanner.nextLine());

                        k = new Konto(number, amount, name, owner);

                        m.addAccount(k);

                        System.out.println("Ditt nya konto ar:\n"
                                    + Metoder.accountToString(k));
            }

            /**
             * Deposits money into an account.
             */
            private static void sattInPengar() {
                        System.out.println("Valkommen till pengainsattningen!");
                        System.out.print("Skriv in kontonummer: ");

                        try {
                                    Konto k = m.findAccount(tbScanner.nextLine());
                                    System.out.println("Konto: " +
                                                Metoder.accountToString(k));
                                    System.out.print
                                                ("Vanligen skriv i hur mycket pengar ni" +
                                                            " vill satta in: ");

                                    double amount = tbScanner.nextDouble();

                                    k.depositAmount(amount);

                                    System.out.println("Du har nu satt in " + amount + "
pengar pa "
                                                + "ditt konto: \n" +
Metoder.accountToString(k));

                        } catch (NoSuchFieldException e){
                                    System.out.println("Finns inget konto med det
numret!");
                                    return;
                        }

            }

            /**
             * Withdraws money into an account.
             */
            private static void taUtPengar() {
                        System.out.println("Valkommen till pengauttagningen!");
                        System.out.print("Skriv in kontonummer: ");
                        try {
                                    Konto k = m.findAccount(tbScanner.nextLine());

```

```
248                                     System.out.println("Konto: " +
249   Metoder.accountToString(k));
250                                     System.out.print
251                                                 ("Vanligen skriv i hur mycket pengar ni
252   vill ta ut: ");
253
254                                     double amount =
255   Double.parseDouble(tbScanner.nextLine());
256
257                                     k.withdraw(amount);
258
259                                     System.out.println("Du har tagit ut " + amount + "
260   pengar pa "
261                                                 + "ditt konto: \n" +
262   Metoder.accountToString(k));
263
264                         } catch (NoSuchFieldException e){
265                                     System.out.println("Finns inget konto med det
266   numret!");
267                                     return;
268                         }
269             }
270
271             /**
272              * Registers transaction.
273              */
274             private static void registreraTransaktion() {
275                         SimpleDateFormat dFormat = new SimpleDateFormat("yyyyMMdd");
276                         Konto source, destination;
277                         Date due;
278                         double amount;
279                         String ocr, not;
280                         Transaktion t;
281
282                         System.out.println("Valkommen att skapa en transaktion!");
283                         try {
284                                     System.out.print("Skriv i avsandarens kontonummer: ");
285                                     source = m.findAccount(tbScanner.nextLine());
286                                     System.out.print("Skriv i mottagarens kontonummer: ");
287                                     destination = m.findAccount(tbScanner.nextLine());
288
289                         } catch (NoSuchFieldException e) {
290                                     System.out.println("Det finns inget konto med det
291   numret!");
292                                     return;
293                         }
294
295                         System.out.print("Skriv vilket datum, i formatet yyyyMMdd, du vill
296   att"
297                                     + " transaktionen ska genomforas: ");
298
299                         try {
300                                     due = dFormat.parse(tbScanner.nextLine());
301                         } catch (ParseException e) {
302                                     System.out.println("Datumet du angav var i ogiltigt
303   format!");
304                                     return;
305                         }
306
307                         System.out.print("Ange summa: ");
308                         try {
```

```
309                                     amount =
310   Double.parseDouble(tbScanner.nextLine().replace(",","."));
311                             } catch(NumberFormatException e) {
312                                     System.out.println("Det dar ar inte ett giltigt
313   tal.");
314                                     return;
315                             }
316
317                             System.out.print("Skriv i ett OCR eller ett medelande :");
318                             ocr = tbScanner.nextLine();
319
320                             if(!m.validOcr(ocr)) {
321                                     System.out.println("Din inmatning registreades som ett
322   meddelande.");
323                             }
324
325                             System.out.println("Lagg till en notering eller lamna faltet
326   blankt: ");
327                             not = tbScanner.nextLine();
328
329                             if(not.trim().length() > 0)
330                                     t = new Transaktion(due, source.getAccountNumber(),
331
332             destination.getAccountNumber(), amount, ocr);
333                             else
334                                     t = new Transaktion(due, source.getAccountNumber(),
335
336             destination.getAccountNumber(), amount, ocr, not);
337
338
339                             m.addTransaction(t);
340                             System.out.println("Transaktion tillagd!\n" + t);
341
342             }
343   }
344
```

```
 1
 2  /** Pgm3.java
 3   *
 4   * Program for archiving and listing transactions
 5   *
 6   * @author Jonathan Skårstedt
 7   * @author Oskar Pålsgård
 8   * @author Magnus Duberg
 9   *
10   * Version 1.0
11   */
12
13  import java.util.*;
14  import java.io.*;
15
16  public class Pgm3{
17              private static Metoder m;
18      private static Scanner tbScanner = new Scanner(System.in);
19
20      /**
21       * Main method
22       *
23       * Generally flows into two methods, to either archive old transactions or
24       * to list logged ones.
25       *
26       * @param args NA: Command line arguments
27       */
28              public static void main(String[] args) {
29          System.out.println("\n-= Pgm3, Hanterar gamla transaktioner =-\n");
30
31                          try {
32                                  m = Metoder.buildMetoder();
33                          } catch (IOException e) {
34                                  System.out.println("Kunde inte öppna angivna filer: "
35                                                                  + e.getMessage());
36                                  return;
37                          }
38
39          String huvudMeny =
40          "=======================================\n" +
41          "== Meny ================================\n" +
42          "=======================================\n" +
43          "1. Arkivera forfallna transaktioner\n" +
44          "2. Lista alla utforda transaktioner\n" +
45          "0. Avsluta\n" +
46          "Ange ditt val: ";
47
48          boolean avsluta = false;
49          while(!avsluta) { //
50              System.out.print(huvudMeny);
51              switch (tbScanner.nextLine()){
52                  case "1":
53                          arkiveraGamlaTransaktioner();
54                          break;
55
56                  case "2":
57                          listaTransaktioner();
58                          break;
59
60                  case "0":
61                          avsluta = true;
```

```
                        break;

                default:
                        System.out.println("Forsok igen! (0-2)");
                        break;
            }
        }
        // Sparar alla ändringar
        try {
            m.saveChanges();
        }catch(IOException e){
            System.out.print("Fel vid sparandet. Exception:" + e);
        }
        // Avslutar programmet
        System.out.println("Avslutar.");
    }


    /**
     * Moves and archives old, non-executed transactions into an archive file.
     *
     * Makes sure the file to archive to doesn't exist, and shows amount of
     * archived transactions
     */
    private static void arkiveraGamlaTransaktioner(){
            File newArchiveFile = null;

                        System.out.print ("Mata in ett nytt filnamn att arkivera till: ");
                        Boolean unikFil = false;
                        do {
                                    String newFileName = tbScanner.nextLine();
                        if (new File("datafiler/"+newFileName).exists()){
                                                System.out.print("Filen finns redan. Mata
in ett nytt filnamn: ");
                        }else{
                                    newArchiveFile = new File(newFileName);
                                    unikFil = true;
                        }
            }while(!unikFil);
            tbScanner.reset();

     // Hämtar datum en vecka tillbaka
            Calendar c = Calendar.getInstance();
        c.add(Calendar.WEEK_OF_YEAR, -1);

     // Försöker arkivera till den nya filen
            System.out.println("Sparar till: " + newArchiveFile);
        try {
            int antalArkiveradeTransaktioner;
            antalArkiveradeTransaktioner = m.archiveTransactions(c.getTime(),
newArchiveFile);
            System.out.println("Arkiverade " + antalArkiveradeTransaktioner + "
transaktioner");
        } catch (IOException e) {
            System.out.println("Kunde inte skriva till arkivfilen!");
        }
    }//slut på arkiveraGamlaTransaktioner


    /**
     * Lists made transactions, sorted by either Date or Account number.
     */
```

```
124       private static void listaTransaktioner(){
125               System.out.print("Vill du lista transaktioner sorterat efter\n" +
126                                       "1. kontonummer  2. datum\n0. Avbryt    Ange ditt val:
127   ");
128           do {
129                       switch (tbScanner.nextLine()){ //Förlåt, String endast java 7+
130                       case "1" : m.printLogs(m.getLogsSortedByAccountNumber()); return;
131                       case "2" : m.printLogs(m.getLogsSortedByDate()); return;
132                       case "0" : return;
133                       default  : System.out.println("Fel inmatning. Försök igen");}
134                   }while (true);
135       }
136   }
```

```
 1
 2  //package weraFinal;
 3
 4  import java.text.SimpleDateFormat;
 5  import java.util.*;
 6  /**
 7   *
 8   * @author Jonathan Skårstedt
 9   * @author Oskar Pålsgård
10   * @author Magnus Duberg
11   *
12   */
13  public class Transaktion {
14              protected Date dueDate;
15              protected String sourceAccount;
16              protected String destinationAccount;
17              protected double amount;
18              protected String ocrMessage;
19              protected String paymentNote;
20
21              private SimpleDateFormat dFormat = new SimpleDateFormat("yyyyMMdd");
22
23              /**
24               * Constructor without notice
25               *
26               * @param dueDate Date of execution
27               * @param sourceAccount Source Account of the transaction
28               * @param destinationAccount
29               * @param amount
30               * @param ocrMessage
31               */
32              public Transaktion(Date dueDate, String sourceAccount,
33                                      String destinationAccount, double amount, String
34  ocrMessage) {
35
36                          setDueDate(dueDate);
37                          setSourceAccount(sourceAccount);
38                          setDestinationAccount(destinationAccount);
39                          setAmount(amount);
40                          setOcrMessage(ocrMessage);
41                          setNotice("");
42              }
43
44              /**
45               * Constructor with notice
46               *
47               * @param d due date of the Transaktion
48               * @param sa Source account of the Transaktion
49               * @param da Destination account of the Transaktion
50               * @param a Amount to transact
51               * @param om OCR message of the Transaktion
52               * @param n Notice of the Transaktion
53               */
54              public Transaktion(Date d, String sa, String da,
55                                      double a, String om, String n) {
56                          setDueDate(d);
57                          setSourceAccount(sa);
58                          setDestinationAccount(da);
59                          setAmount(a);
60                          setOcrMessage(om);
61                          setNotice(n);
```

```
62              }
63
64              /**
65               * Returns a Transaktion object as a String representation
66               * @return A formatted string of the transaction
67               */
68              public String toString(){
69                              return "Transaction\n\t"
70                                                              + getDueDate() + "\t"
71                                                              + getSourceAccount() + "\t"
72                                                              + getDestinationAccount() + "\t"
73                                                              + getAmount() + "\t"
74                                                              + getOcrMessage() + "\t"
75                                                              + getNotice() + "\n\n";
76              }
77
78              /**
79               * Formats a Transaktion for file output
80               *
81               * @return A string representing a Transaktion object in log form
82               */
83              public String toFileString(){
84                              String out = dFormat.format(getDueDate()) + "#" +
85      getSourceAccount()
86                                              + "#" + getDestinationAccount() + "#" + getAmount() +
87      "#"
88                                              + getOcrMessage();
89
90                              if(paymentNote.trim().length() > 0)
91                                              out += ";" + paymentNote;
92
93                              return out;
94              }
95
96              public static void traverseMonitoredAccounts() throws Exception {
97                              throw new Exception("not implemented");
98              }
99
100             // Get methods
101             public Date getDueDate(){return dueDate; }
102             public String getSourceAccount(){return sourceAccount; }
103             public String getDestinationAccount(){return destinationAccount; }
104             public double getAmount(){return amount; }
105             public String getOcrMessage(){return ocrMessage; }
106             public String getNotice(){return paymentNote; }
107
108             // Set methods
109             public void setDueDate(Date d){dueDate = d; }
110             public void setSourceAccount(String sa){sourceAccount = sa; }
111             public void setDestinationAccount(String da){destinationAccount = da; }
112             public void setAmount(double a){amount = a; }
113             public void setOcrMessage(String om){ocrMessage = om; }
114             public void setNotice(String n){paymentNote = n; }
115     }
```

```
1
2   import java.text.SimpleDateFormat;
3   import java.util.*;
4
5   class GjordTransaktion extends Transaktion {
6
7               public static enum TransactionType {TRANSACTION, WITHDRAWAL,
8                                       DEPOSIT, INVALID};
9
10  /* derived from Transaktion
11   *          protected Date dueDate;
12              protected String sourceAccount;
13              protected String destinationAccount;
14              protected double amount;
15              protected String ocrMessage;
16              protected String notice;
17   */
18              private String transactionNote;
19              private Date transactionDate;
20
21              private SimpleDateFormat dFormat = new SimpleDateFormat("yyyyMMdd");
22
23  /**
24   * analyzeTransactionType()
25   * @param raw
26   * @return
27   */
28              public static TransactionType analyzeTransactionType(String raw) {
29                          /* Withdrawal
30                          * 0: Status, 1: Transaction date, 2: Planned date,
31                          * 3: Source account, 4: KONTANTER
32                          * 5: Amount to deposit, 6: OCR Message */
33                          if(raw.matches(".+;[0-9]{8}#[0-9]{8};[0-9]+-[0-9]+;KONTANTER;" +
34                                                  "[0-9]+,[0-9]{1,2};.+")) {
35                                      return TransactionType.WITHDRAWAL;
36                          }
37
38                          /* Deposit
39                             0: Status, 1: Transaction date, 2: Planned date,
40                             3: "KONTANTER", 4: Destination account
41                             5: Amount to deposit, 6: OCR Message */
42                          else if(raw.matches(".+;[0-9]{8}#[0-9]{8};KONTANTER;[0-9]+-[0-
43  9]+;" +
44                                                  "[0-9]+,[0-9]{1,2};.+")) {
45                                      return TransactionType.DEPOSIT;
46                          }
47
48                          /* Transaction
49                             0: Status, 1: Transaction date, 2: Planned date,
50                             3: Source account, 4: Destination account,
51                             5: Amount to deposit, 6: OCR message */
52                          else if(raw.matches(".+;[0-9]+#[0-9]+;[0-9]+-[0-9]+;[0-9]+-[0-
53  9]+;" +
54                                                  "[0-9]+,[0-9]{1,2};.+")) {
55                                      return TransactionType.TRANSACTION;
56                          }
57
58                          /* Non-valid split */
59                          else {
60                                      return TransactionType.INVALID;
61                          }
```

```
62              }
63
64              /**
65               * Constructor
66               *
67               */ /*@param transType*/ /**
68               * @param transNote
69               * @param transDate
70               * @param dueDate
71               * @param sourceAccount
72               * @param destinationAccount
73               * @param amount
74               * @param ocrMessage
75               * @param paymentNote
76               */
77              public GjordTransaktion(String transNote, Date transDate,
78                                      Date dueDate, String sourceAccount, String
79  destinationAccount,
80                                      double amount, String ocrMessage, String paymentNote){
81                          super(dueDate, sourceAccount, destinationAccount,
82                                                  amount, ocrMessage, paymentNote);
83
84                          setTransactionDate(transDate);
85                          setTransactionNote(transNote);
86              }
87
88              /** Construktor that takes a Transaction object
89               * @param transNote String
90               * @param transDate Date
91               * @param trans Transaktion
92               */
93              public GjordTransaktion(String transNote,Date transDate,Transaktion trans){
94                          super(trans.dueDate, trans.sourceAccount,
95  trans.destinationAccount,
96                                                  trans.amount, trans.ocrMessage,
97  trans.paymentNote);
98                          setTransactionDate(transDate);
99                          setTransactionNote(transNote);
100             }
101
102
103
104
105             /**
106              *
107              */
108             public String toFileString(){
109                         String ret = "";
110
111                         ret += transactionNote + ";" + dFormat.format(transactionDate) +
112 "#"
113                                     + dFormat.format(dueDate) + ";"
114                                     + sourceAccount + ";" + destinationAccount + ";"
115                                     + Double.toString(amount).replace(".", ",") + ";" +
116 ocrMessage;
117
118                         if(paymentNote.length() > 0){
119                                     ret += ";" + paymentNote;
120                         }
121                         return ret;
122             }
123
```

```
124                 // Set methods
125     //          public void setType(TransactionType t){type = t;}
126                 public void setTransactionNote(String tn){transactionNote = tn;}
127                 public void setTransactionDate(Date ex){transactionDate = ex;}
128                 // Get methods
129     //          public TransactionType getType(){return type;}
130                 public String getTransactionNote(){return transactionNote;}
131                 public Date getTransactionDate(){return transactionDate;}
132
133     }
```

```
 1
 2  import java.util.*;
 3  import java.io.*;
 4  import java.text.*;
 5
 6  /**
 7   *
 8   * @author Jonathan Skårstedt
 9   * @author Oskar Pålsgård
10   * @author Magnus Duberg
11   *
12   */
13  public class Metoder {
14              private File accountFile;
15              private File transactionLogFile;
16              private File surveillanceFile;
17
18              private SimpleDateFormat dFormat = new SimpleDateFormat("yyyyMMdd");
19              private static String dataPath = "datafiler/";
20
21              public Konto[] accounts; // Konton
22              public ArrayList<Transaktion> transactions; // Bevakning
23              public ArrayList<GjordTransaktion> transactionLog; // Gjorda transaktioner
24              public ArrayList<String> splitLine; // ???
25
26              /**
27               * Constructor for the Metoder object
28               * Reads all data files and loads them into object
29               *
30               * @param accountPath Account file path
31               * @param logPath Log file path
32               * @param surveillancePath Surveillance file path
33               * @throws FileNotFoundException If files are not found
34               */
35              public Metoder(String accountPath, String logPath,
36                                      String surveillancePath) throws FileNotFoundException{
37                          accountFile = new File(accountPath);
38                          transactionLogFile = new File(logPath);
39                          surveillanceFile = new File(surveillancePath);
40
41                          readAccounts();                     //ladda _Konton.txt
42                          readTransactions();     //ladda _Bevakning.txt
43                          readLog();                          //ladda
44  _GjordaTransaktioner.txt
45              }
46
47              /** Reads the log and loads the to Metoder
48               * @throws FileNotFoundException if file isn't found
49               */
50              public void readLog() throws FileNotFoundException{
51                          /*DEBUG*/ System.out.print("Executing readLog:");
52
53                          transactionLog = new ArrayList<GjordTransaktion>();
54                          Scanner logFileScanner = new Scanner(transactionLogFile);
55
56                          GjordTransaktion gjordTrans;
57                          GjordTransaktion.TransactionType transType;
58                          int lineNumber = 0;
59
60              // För varje rade ur transaktionsfilen
61                          while(logFileScanner.hasNextLine()) {
```

```
                                    lineNumber += 1;
                                    String line = logFileScanner.nextLine();
                                    transType =
GjordTransaktion.analyzeTransactionType(line);
                                    String[] splittedLine = line.split(";|#");

                          // Försök skapa ett GjordTransaktions-objekt
                                    try {
                                              gjordTrans = new
GjordTransaktion(splittedLine[0], // trans.not

              dFormat.parse(splittedLine[1]), // trans.datum

              dFormat.parse(splittedLine[2]), // önskat datum
                                                                        splittedLine[3],
splittedLine[4], // käll- & dest.konto

              parseSweDouble(splittedLine[5]), // belopp
                                                                        splittedLine[6], //
ocrMeddelande
                                                                        "");
                                    } catch (NumberFormatException e) {
                                              System.out.println("NumberFormatException
at row("+
                                                                        lineNumber+"): " +
e.getMessage());
                                              break;
                                    } catch (ParseException e) {
                                              System.out.println("Error reading
row("+lineNumber+"): " +
                                                                        e.getMessage());
                                              break;
                                    } // end try

                          // Lägg till betalningsnotering om det finns
                                    if(splittedLine.length > 7)
                                              gjordTrans.setNotice(splittedLine[7]);

                          // Avryter programmet om inte klarat regexp i
analyzeTransactionType
                                    switch (transType){
                                    case DEPOSIT:break;
                                    case WITHDRAWAL:break;
                                    case TRANSACTION:break;
                                    case INVALID:
                                              System.out.println("Invalid
line("+lineNumber+") in logfil: ");
                                              System.out.println("Line nr = "+line);
                                              System.exit(0);
                                              break;
                                    }
                          // Lägg till gjord transaktion i gjorda transaktioner-objektet
                                    transactionLog.add(gjordTrans);

                          }// end while
/*DEBUG*/ System.out.println(" laddat " + transactionLog.size() + " fran " +
transactionLogFile);
                          logFileScanner.close();
              }

              /** Validate an OCR number according to the Luhn algorithm
               * http://en.wikipedia.org/wiki/Luhn_algorithm
```

```
124              * @param ocrNumber OCR number to validate
125              * @return true on valid OCR number
126              */
127             public boolean validOcr(String ocrNumber) {
128                         // Kollar f�rst om det bara �r siffror
129                     try {
130                         Double.parseDouble(ocrNumber);
131                     } catch (NumberFormatException e) {
132                         return false;
133                     }
134                 // Kollar om l�ngden �r mellan 2 och 15 tecken
135                     if (2 > ocrNumber.length() || ocrNumber.length() > 15)
136                         return false;
137
138                         int checksum = 0;
139                         boolean alt = false;
140
141                         for (int i = ocrNumber.length() - 1; i >= 0; i--){
142                                     int n = Integer.parseInt(ocrNumber.substring(i, i +
143 1));
144
145                                     if(alt) {
146                                                 n *= 2;
147
148                                                 if(n > 10)
149                                                             n++;
150                                     }
151
152                                     checksum += n % 10;
153                                     alt = !alt;
154                         }
155
156                         return (checksum % 10 == 0);
157             }
158
159             /** Adds a Transaktion to the transactions list
160              * @param t Transaktion to add
161              */
162             public void addTransaction(Transaktion t){
163                         transactions.add(t);
164             }
165
166             /** Translates a swedish double precision string representation of a digit
167              * to a double. String must be in "x,y" form.
168              * @param in Swedish String double to convert to double
169              * @return The String transformed to a double
170              * @throws NumberFormatException on not being in "x,y" form
171              */
172             public double parseSweDouble(String in) throws NumberFormatException {
173                         return Double.parseDouble(in.replace(",", "."));
174             }
175
176             /** Läser transaktioner från bevakningsfilen och laddar dem till en ArrayList
177              */
178             public void readTransactions(){
179 /*DEBUG*/ System.out.print("Executing readTransactions: ");
180
181                         Scanner transScanner;
182                         String[] transStrings;
183
184                         transactions  = new ArrayList<Transaktion>();
185                         try {
```

```
186                                     transScanner = new Scanner(surveillanceFile);
187
188                                     while(transScanner.hasNextLine()) {
189                                                 transStrings =
190   transScanner.nextLine().split("#");
191
192                                                 try {
193                                                             if(transStrings.length == 5) {
194
195               transactions.add(new Transaktion(dFormat.parse(transStrings[0]),
196
197               transStrings[1], transStrings[2],
198
199               parseSweDouble(transStrings[3]), transStrings[4]));
200                                                             }
201                                                             else {
202
203               transactions.add(new Transaktion(dFormat.parse(transStrings[0]),
204
205               transStrings[1], transStrings[2],
206
207               parseSweDouble(transStrings[3]), transStrings[4],
208
209               transStrings[5]));
210                                                             }
211                                                 } catch (ParseException e) {
212                                                             System.out.println("Not a
213   valid date, ignoring: "
214                                                                                                 +
215   e.getMessage());
216                                                 }
217                                     }
218   /*DEBUG*/ System.out.print("laddat " + transactions.size() + " fran " + surveillanceFile);
219                                     transScanner.close();
220                           }catch (FileNotFoundException e){
221                                     System.out.println("fil kunde inte hittas.");
222                           }
223                           System.out.println();
224               }
225
226               /** Reads an account file; sorts and parses the data to an Array.
227                *                    Jumps over empty lines
228                * @throws FileNotFoundException if the account file isn't found
229                */
230               public void readAccounts() throws FileNotFoundException {
231   /*DEBUG*/ System.out.print("Executing readAccounts: ");
232
233                           accounts = new Konto[400];
234                           Scanner accountFileScanner;
235                           String[] splittedAccountLine;
236                           String accountLine;
237
238                           accountFileScanner = new Scanner(accountFile);
239
240                           int numberOfAccounts = 0;
241                           for(int i = 0; accountFileScanner.hasNextLine(); i++){
242                                     accountLine = accountFileScanner.nextLine();
243                                     if(accountLine.trim().equals(""))
244                                                 continue;
245
246                                     splittedAccountLine = accountLine.split("##");
247
```

```
248                                     accounts[i] = new Konto(splittedAccountLine[0],
249
250             parseSweDouble(splittedAccountLine[1]),
251                                                             splittedAccountLine[2],
252                                                             splittedAccountLine[3]);
253                                     numberOfAccounts = i;
254                             }
255     /*DEBUG*/ System.out.println("laddat " + numberOfAccounts + " fran " + accountFile);
256                             accountFileScanner.close();
257
258                             // sorting the array
259                             Konto temp;
260                             boolean sorted = true;
261                             while(sorted) {
262                                     sorted = false;
263                                     for(int i = numberOfAccounts; i > 0 && i <=
264     numberOfAccounts; i--){
265                                                     if(accounts[i].getAccountNumber()
266
267             .compareTo(accounts[i-1].getAccountNumber()) < 0){
268
269                                                             sorted = true;
270                                                             temp = accounts[i];
271                                                             accounts[i] = accounts[i-1];
272                                                             accounts[i-1] = temp;
273
274                                                     }
275                                     }
276                             }
277             }
278
279             /**
280              * Translate a Konto to a formatted string
281              * @param k Konto to format
282              * @return If found returns a formatted string, else a Konto object
283              */
284             public static String accountToString(Konto k) {
285                             if(k == null)
286                                     return "";
287
288                             return String.format("%-20s %-15s %20.2f %20s",
289     k.getAccountName(),
290                                                     k.getAccountNumber(),
291     k.getAvailableAmount(),
292                                                     k.getOwnerName());
293             }
294
295             /**
296              * Traverses the transaction list. Executes and removes transactions
297              *                       between two dates.
298              *
299              * @param after The date where after to remove a certain transaction
300              * @param before The date where before to remove a certain transaction
301              */
302             public void executeAllTransactionsBetween(Date after, Date before) {
303
304                             System.out.println("Executing executeAllTransactionsBetween:");
305
306                             for(Iterator<Transaktion> it = transactions.iterator();
307     it.hasNext();) {
308                                     Transaktion t = it.next();
309
```

```
                                    if(t.dueDate.before(before) && t.dueDate.after(after))
{
                                                try {
                                                            executeTransaction(t);
                                                }catch( Exception e){
                                                            System.out.println("Exception:
" + e);
                                                }
                                                it.remove();
                                    }
                        }
            System.out.println(" Antal transaktioner efter: "+transactions.size());
            }

            /**
             * Archives older transactions to file
             * @param olderThan Only archive transactions older than this
             * @param archiveFile file where to save the archive
             * @throws IOException
             */
            public int archiveTransactions(Date olderThan, File archiveFile)
                                    throws IOException{

                        BufferedWriter archiveWriter
                        = new BufferedWriter(new FileWriter(dataPath +
archiveFile.getPath()));

            //
                        int antalTransaktioner = transactions.size();
                        for(Iterator<Transaktion> it = transactions.iterator();
it.hasNext();) {
                                    Transaktion t = it.next();
                                    if(t.dueDate.before(olderThan)) { // Alla gamla
transaktioner
                                                archiveWriter.write(t.toFileString() +
"\r\n"); // arkiveras
                                                it.remove(); // och tas bort från
transaktions-listan
                                    }
                        }
                        archiveWriter.close();
                        // Returnerar antalet arkiverade transaktioner
                        return antalTransaktioner - transactions.size();
            }

            /** Executes a Transaktion - withdraws and deposit as described in the
             *                     transaction file
             * @param trans Transaktion to execute
             */
            public void executeTransaction(Transaktion trans){

                        try {
                                    Konto source = findAccount(trans.getSourceAccount());
                                    Konto destination =
findAccount(trans.getDestinationAccount());

                                    if (trans.amount <= source.getAvailableAmount()){ //
om tillräckligt
                                                source.withdraw(trans.getAmount()); //
flyttas pengarna

            destination.depositAmount(trans.getAmount()); // till dest.konto
```

```
372                                             appendToLog("OK", trans);
373                                     }
374                                     else { // transaktionen genomförs inte
375                                             //
376                                     }
377                             } catch (Exception e) {
378                                     System.out.println("Couldn't execute transaction '"
379                                                     +trans.toFileString()+"'");
380                             }
381                     }
382
383                     /** Checks if an account exists
384                      * @param account account number to check if it exists
385                      * @return true if account exists, else false
386                      */
387                     public boolean accountExists(String account){
388                             try {
389                                     findAccount(account);
390                                     return true;
391                             } catch (NoSuchFieldException e) {
392                                     return false;
393                             }
394
395
396                     }
397                     /**
398                      * Finds an account by binary search
399                      *
400                      * @param account Account string to search for
401                      * @return The found account
402                      */
403                     public Konto findAccount(String account) throws NoSuchFieldException{
404                             int size;
405                             for(size = 0; accounts[size] != null; size++);
406
407                             int max = size;
408                             int min = 0;
409                             int pos = max / 2;
410
411                             String current;
412
413                             while(!accounts[pos].getAccountNumber().equals(account)) {
414                                     current = accounts[pos].getAccountNumber();
415
416                                     if(current.compareTo(account) > 0)
417                                             max = pos;
418                                     else
419                                             min = pos;
420
421                                     if(pos == (max - min) / 2 + min)
422                                             throw new NoSuchFieldException
423                                                     ("Couldn't find account with
424     number '"+account+"'");
425
426                                     pos = (max - min) / 2 + min;
427                             }
428                             return accounts[pos];
429                     }
430
431                     /**
432                      * Lägger till en GjordTransaktion till transaktions-loggen i minnet
433                      * @param status
```

```
434                * @param trans
435                */
436               public void appendToLog(String status, Transaktion trans){
437                           GjordTransaktion gjordTrans =
438                                                       new GjordTransaktion(status, new Date(),
439   trans);
440
441                           transactionLog.add(gjordTrans);
442               }
443
444               /**
445                * Returnerar en kopia av konton
446                * @return Konto[]-lista
447                */
448               public Konto[] getAccounts(){
449                           return accounts.clone();
450               }
451
452               /**
453                * Save current changes to given files.
454                * @throws IOException
455                */
456               public void saveChanges() throws IOException{
457               System.out.println("Executing saveChanges:");
458
459   /*DEBUG*/ System.out.println(" Kontofil: "+accountFile.getPath());
460   /*DEBUG*/ System.out.println(" Gjorda transaktioner: "+transactionLogFile.getPath());
461   /*DEBUG*/ System.out.println(" Bevakningsfil: "+surveillanceFile.getPath());
462
463               // Skapa lista på datafilerna
464                           File[] allFiles = {accountFile, transactionLogFile,
465   surveillanceFile};
466               // För varje fil
467                           for(File eachFile: allFiles) {
468                           // Hoppa över transaktionsloggen
469                                       if(eachFile.equals(transactionLogFile)) continue;
470
471                           // Skapar backupfil
472                                       File backupFile = new
473   File(eachFile.getPath().substring(0,
474
475               eachFile.getPath().lastIndexOf(".")) + ".bak");
476   /*DEBUG*/   System.out.print(" Backupfil: " + backupFile);
477                           // Tar bort den om redan finns
478                                       if(backupFile.exists()){
479   /*DEBUG*/               System.out.print(", tar bort gamla, ");
480                                                   backupFile.delete();
481                                       }
482                           // Byter namn till backupfil
483                                       if(eachFile.renameTo(backupFile)){
484                                                   System.out.println("bytt namn.");
485                                       }
486                           }
487
488               // Skriver konton till fil
489                           BufferedWriter accountWriter = new BufferedWriter(new
490   FileWriter(accountFile));
491                           for(Konto acc : accounts){
492                                       if(acc == null){
493                                                   break;
494                                       }
495                                       else{
```

```
                                                accountWriter.write(acc.getAccountNumber()
+ "##"
                                                                                +
acc.getAvailableAmount() + "##"
                                                                                +
acc.getAccountName() + "##"
                                                                                +
acc.getOwnerName() + "\r\n");
                                        }
                        }
                        // Skriver återstående transaktioner till bevakningsobjektet
                        BufferedWriter surveillanceWriter = new BufferedWriter(new
FileWriter(surveillanceFile));
                        for(Transaktion t : transactions){
                                surveillanceWriter.write(t.toFileString() + "\r\n");
                        }
                        // Skriver utförda transaktioner till transaktionsloggen
                        BufferedWriter logWriter = new BufferedWriter(new
FileWriter(transactionLogFile));
                        for(GjordTransaktion gt : transactionLog) {
/*DEBUG/      System.out.println("gjordTrans.toFileString: "+gt.toFileString());
            /*DEBUG*/
                                logWriter.write(gt.toFileString() + "\r\n");
                        }

                        // Sparar och stänger datafilerna
                        accountWriter.close();
                        surveillanceWriter.close();
                        logWriter.close();
            }

            /**
             * Lägger till ett konto och sorterar kontolistan
             * @param k
             */
            public void addAccount(Konto k){
                        int size;
                        for(size = 0; accounts[size] != null; size++);
                        accounts[size++] = k;

                        // sorting the array
                        Konto temp;
                        boolean sorted = true;
                        while(sorted){
                                sorted = false;
                                for(int i = size - 1; i > 0 && i <= size; i--){
                                        if(accounts[i].getAccountNumber()

            .compareTo(accounts[i-1].getAccountNumber()) < 0){

                                                                sorted = true;
                                                                temp = accounts[i];
                                                                accounts[i] = accounts[i-1];
                                                                accounts[i-1] = temp;

                                        }
                                }
                        }
            }


            /**
```

```
558                 * Creates a method object for handling bank business
559                 * @return a freshly baked Metoder object
560                 * @throws FileNotFoundException
561                 */
562                public static Metoder buildMetoder() throws IOException{
563
564                            BufferedReader in
565                            = new BufferedReader(new InputStreamReader (System.in));
566                            String temp, accounts, log, survey;
567
568                            String defaultAccountsFilePath = dataPath + "_Konton.txt";
569                            String defaultLogFilePath = dataPath +
570   "/_GjordaTransaktioner.txt";
571                            String defaultSurveillanceFilePath = dataPath + "/_Bevakning.txt";
572
573                            System.out.print("Ange kontofil (default="+
574   defaultAccountsFilePath +"): ");
575                            temp = in.readLine();
576                            if(temp.trim().length() > 0) {
577                                        accounts = temp;
578                            } else {
579                                        accounts = defaultAccountsFilePath;
580                            }
581
582                            System.out.print("Ange loggfil (default="+defaultLogFilePath+"):
583   ");
584                            temp = in.readLine();
585                            if(temp.trim().length() > 0) {
586                                        log = temp;
587                            } else {
588                                        log = defaultLogFilePath;
589                            }
590
591                            System.out.print("Ange bevakningsfil
592   (default="+defaultSurveillanceFilePath+"): ");
593                            temp = in.readLine();
594                            if(temp.trim().length() > 0)
595                                        survey = temp;
596                            else {
597                                        survey = defaultSurveillanceFilePath;
598                            }
599
600   /*DEBUG/
601                            System.out.println("Använder: " + new File(".").getAbsolutePath()
602   +
603                                                                                    "\nKontofil: \t"
604   + accounts +
605                                                                                    "\nLogfil: \t" +
606   log +
607                                                                                    "\nBevakningsfil:
608   \t" + survey);
609   /*DEBUG*/
610                            return new Metoder(accounts, log, survey);
611                }
612
613                /**
614                 * Returnerar en ArrayList med utförda transaktioner sorterade efter datum
615                 * @return ArrayList<GjordTransaktion>
616                 */
617                public ArrayList<GjordTransaktion> getLogsSortedByDate(){
618                            int size = transactionLog.size();
619                            ArrayList<GjordTransaktion> gTransToSort = transactionLog;
```

```
620
621                         // sorting the ArrayList
622                         GjordTransaktion tempGT;
623                         boolean sorted = true;
624                         while(sorted){
625                                     sorted = false;
626                                     for(int i = size - 1; i > 0 && i <= size; i--){
627
628             if(gTransToSort.get(i).getTransactionDate()
629
630             .compareTo(gTransToSort.get(i-1).getTransactionDate()) < 0){
631
632                                                                 sorted = true;
633                                                                 tempGT = gTransToSort.get(i);
634
635             gTransToSort.set(i,gTransToSort.get(i-1));
636                                                                 gTransToSort.set(i-1,tempGT);
637
638                                                     }
639                                     }
640                         }
641                         return gTransToSort;
642             }
643
644             /** Get logs sorted by account number
645              * @return ArrayList of gjordatransaktioner
646              */
647             public ArrayList<GjordTransaktion> getLogsSortedByAccountNumber(){
648                         System.out.println();
649                         int size = transactionLog.size();
650                         ArrayList<GjordTransaktion> gTransToSort = transactionLog;
651
652                         // sorting the ArrayList
653                         GjordTransaktion tempGT;
654                         boolean sorted = true;
655                         while(sorted){
656                                     sorted = false;
657                                     for(int i = size - 1; i > 0 && i <= size; i--){
658                                                 if(gTransToSort.get(i).getSourceAccount()
659
660             .compareTo(gTransToSort.get(i-1).getSourceAccount()) < 0){
661
662                                                                 sorted = true;
663                                                                 tempGT = gTransToSort.get(i);
664
665             gTransToSort.set(i,gTransToSort.get(i-1));
666                                                                 gTransToSort.set(i-1,tempGT);
667
668                                                     }
669                                     }
670                         }
671                         return gTransToSort;
672             }
673             /**
674              * Skriver ut sorterad och formaterad lista över utförda transaktioner
675              * @param transList
676              */
677             public void printLogs(ArrayList<GjordTransaktion> transList){
678                         System.out.println(
679 "OK Utford   Planerad Kallkonto    Dest.konto   Summa     OCR/Meddelande  Not\n"+
680 "-- -------- -------- ------------ ------------ --------- --------------- -------");
681                         for (GjordTransaktion gT : transList){
```

```
                                    System.out.println(String.format(
                                                                    "%1$-2s %2$tY%2$tm%2$td
%3$tY%3$tm%3$td %4$-12s %5$-12s %6$9.2f %7$-15s %8$-10s",
                                                                    gT.getTransactionNote(),
gT.getTransactionDate(),
                                                                    gT.getDueDate(),
gT.getSourceAccount(),
                                                                    gT.getDestinationAccount(),
gT.getAmount(),
                                                                    gT.getOcrMessage(),
gT.getNotice()));
                            }
                }
}
```