Problem 1

Create a program handling bank accounts. The program reads a file consisting of lines of three kinds:

1. Representing creation of a new account:
   ```
   JohnN John Novak 2000
   ```
   New account is created with identifier JohnN, owner John Novak and initial deposit 2000.
2. Representing deposit/withdrawal:
   ```
   JohnN 500
   ```
   or
   ```
   JohnN -200
   ```
   The balance of the account with identifier JohnN is increased by 500 (decreased by 200, if the amount is negative).
3. Representing a transfer between accounts:
   ```
   JohnN MaryM 500
   ```
   means that John is sending money to Marry, so the John's balance is decreased by 500 and that of Mary increased by 500.

The data in any line may be erroneous (for example, its format does not correspond to any of the formats described above, or there are insufficient funds on an account for a given withdrawal or transfer etc.); if this happens, the offending line is ignored and an appropriate information is appended to a log file (the offending line, its number in the input file and a description of the reason of the failure).
The functionality of the program is as follows:

- After reading a line of the first kind, the program creates an object of immutable type **Person** (with first and last names) and an object of class **Account** with fields id (a **String** with an account identifier), owner (of type **Person**), balance (**int** – the fourth item of the line denotes the initial deposit), and list – list of transactions (of type **List<Transaction>**). Created objects of type **Account** are then inserted into an appropriate collection, for example a map with account identifiers as keys and these objects as values.
  Transaction are represented by objects of class **Transaction** with a timestamp – for example a **long** returned by
  ```
  System.currentTimeMillis();
  ```
  and data describing the transaction: identifiers of the source and target accounts, amount transferred, and a type of this transaction.
  If the initial deposit is non-positive, or an account with given identifier already exists, the line is considered erroneous.
- After reading a line of the second kind, the balance of the account involved is modified; if the amount of a withdrawal exceeds the balance or the id does not correspond to an existing account, the line is considered erroneous.

- After reading a line of the third kind, the balance of the two accounts involved are modified: if the amount of the transfer exceeds accessible funds of the sender or at least one of the identifiers does not correspond to an existing account, the line is considered erroneous.

Types of transactions can be represented by an **enum** or, in a somewhat old fashioned way, by constants defined in class **Transaction**

```
class Transaction {
    public static final int INIT_DEPOS = 0;
    public static final int DEPOSIT    = 1;
    public static final int WITHDRAWAL = 2;
    public static final int TRANS_FROM = 3;
    public static final int TRANS_TO   = 4;
    private static final String[] opTypes =
        {
            "Init depos ", "Deposit    ", "Withdrawal ",
            "Trans. from", "Trans. to  "
        };
    // ...
}
```

All classes should override the **toString** method.

The following **main** function in class **Bank**

download *Bank.java*

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.Map;
import static java.nio.charset.StandardCharsets.UTF_8;

public class Bank {
    public static void main (String[] args) {
        String fIn  = "Bank.dat";
        String fErr = "Bank.err";
        Map<String,Account> accs = readData(fIn,fErr);

        for (Map.Entry<String,Account> e : accs.entrySet())
            System.out.print(e.getValue());

        try {
            String errLog = new String(
                Files.readAllBytes(Paths.get(fErr)), UTF_8);
            System.out.println("\nContent of " +
                    "\"Bank.err\" follows:\n");
            System.out.println(errLog);
        } catch(IOException e) {
```

```
                System.out.println("Problems with error log");
                return;
            }
        }
        // ...
    }
```

invokes the function **readData** which reads the input file (here ***Bank.dat***), writes information about errors into a log file (here ***Bank.err***) and returns a map of type **Map<String,Account>**. Then, information about all accounts is printed (with lists of transactions). Finally, the log file is read in and printed.

For example, for the data file

```
MaryB Mary Bear 2000
JohnD John  Doe 1000
MaryB Mary Coe   200
JohnD MaryB 1100
JohnD MaryB  900
MaryB 500
MaryB JohnD 50
JohnD -50
```

the program should print something like

```
*** Acc MaryB (Mary Bear). Balance: 3350. Transactions:
   2000: Init depos
    900: Trans. to   this account from John Doe (JohnD)
    500: Deposit
    -50: Trans. from this account   to John Doe (JohnD)
*** Acc JohnD (John Doe). Balance: 100. Transactions:
   1000: Init depos
   -900: Trans. from this account   to Mary Bear (MaryB)
     50: Trans. to   this account from Mary Bear (MaryB)
    -50: Withdrawal

Content of "Bank.err" follows:

Line   3: MaryB Mary Coe   200
   Error: Account already exists
Line   4: JohnD MaryB 1100
   Error: Insufficient funds
```
