Please complete all parts. All parts are worth equal marks.

Write a Java class called `Bank` that is responsible for a collection of customer bank accounts (`Map<Integer, Account>` where the integer refers to the account number of the account) and uses a `LinkedBlockingQueue` object, as described below, to maintain a collection of bank account transactions to be processed on these accounts. Please see the Java API documentation for details on how to use the `LinkedBlockingQueue` class. You should represent bank accounts using the `Account` class that is provided to you on Canvas. Note that you'll need to adapt this class so that it can be used in concurrent applications. You should represent transactions using the `Transaction` class that is provided to you on Canvas. Note that the bank only deals in the currency EUR.

Your `Bank` class should contain methods for the following:

- adding an account to the `Bank`
- getting an account given its account number
- submitting a `Transaction` to the queue for processing
- getting the next `Transaction` from the queue for processing
- printing the accounts' details (account number and balance)
- getting a collection of account numbers

Write classes that represent the following two kinds of threads:

`TransactionProcessor`:

- created with a name and a `Bank` instance
- takes a transaction from the bank and processes it by depositing/withdrawing from the appropriate account
- keeps a tally of how many withdrawals and how many deposits it has made
- sleeps for a random amount of time (between 0 and 1 seconds) between the processing of transactions
- finishes executing once the queue has been closed or if it has waited 5 seconds for a new transaction from the queue without receiving one.
- when it finishes executing, it prints to the console its name and the number of deposits and withdrawals it has processed.

`RandomTransactionGenerator`:

- created with a `Bank` instance
- randomly generates deposit and withdrawal transactions of up to EUR 10,000 for random accounts in the bank and submits them to the bank queue for processing
- sleeps for a random amount of time (between 0 and 1 seconds) between the generation of transactions.
- one it has been terminated, it inserts an *end-of-stream* (or "Poison pill") object to the bank queue to indicate that it is closed.

Write a class with a main method (or include a main method in the `Bank` class) that does the following:

- declares and instantiates a `Bank` object
- creates and adds three `Account` instances to the bank with different starting balances

- declares and instantiates two `TransactionProcessor` threads and one `RandomTransactionGenerator` thread
- executes the threads using a thread pool (`ExecutorService`) and waits for them to complete
- After 10 seconds, explicitly shuts down the `RandomTransactionGenerator` thread
- prints out the details of the accounts after the transactions have finished

Sample output:

```
TPT1 is processing a withdrawal of EUR -6026.66 from 12347.
TPT2 is processing a deposit of EUR 4195.47 to 12345.
TPT1 is processing a deposit of EUR 4200.62 to 12347.
TPT2 is processing a deposit of EUR 8885.09 to 12347.
TPT1 is processing a withdrawal of EUR -9874.27 from 12345.
TPT2 is processing a withdrawal of EUR -3291.50 from 12347.
TPT1 is processing a deposit of EUR 8344.35 to 12346.
TPT2 is processing a deposit of EUR 593.43 to 12346.
TPT2 is processing a withdrawal of EUR -7646.39 from 12345.
TPT1 is processing a deposit of EUR 9323.29 to 12345.
Account 12345 has a balance of EUR 21404.50. Account
12346 has a balance of EUR 9901.38.
Account 12347 has a balance of EUR 21661.47.
Transaction generator terminated.
TPT2 finished processing 9 transactions, including 5 deposits, and 4 withdrawals. TPT1
finished processing 9 transactions, including 5 deposits, and 4 withdrawals.
```