# 5903. Simple Bank System

My Submissions (/contest/weekly-contest-263/problems/simple-bank-system/submissions/) Back to Contest (/contest/weekly-contest-263/)

| | |
|---|---|
| User Accepted: | 0 |
| User Tried: | 0 |
| Total Accepted: | 0 |
| Total Submissions: | 0 |
| Difficulty: | Medium |

You have been tasked with writing a program for a popular bank that will automate all its incoming transactions (transfer, deposit, and withdraw). The bank has `n` accounts numbered from `1` to `n`. The initial balance of each account is stored in a **0-indexed** integer array `balance`, with the `(i + 1)th` account having an initial balance of `balance[i]`.

Execute all the **valid** transactions. A transaction is **valid** if:

- The given account number(s) are between `1` and `n`, and
- The amount of money withdrawn or transferred from is **less than or equal** to the balance of the account.

Implement the `Bank` class:

- `Bank(long[] balance)` Initializes the object with the **0-indexed** integer array `balance`.
- `boolean transfer(int account1, int account2, long money)` Transfers `money` dollars from the account numbered `account1` to the account numbered `account2`. Return `true` if the transaction was successful, `false` otherwise.
- `boolean deposit(int account, long money)` Deposit `money` dollars into the account numbered `account`. Return `true` if the transaction was successful, `false` otherwise.
- `boolean withdraw(int account, long money)` Withdraw `money` dollars from the account numbered `account`. Return `true` if the transaction was successful, `false` otherwise.

**Example 1:**

```
Input
["Bank", "withdraw", "transfer", "deposit", "transfer", "withdraw"]
[[[10, 100, 20, 50, 30]], [3, 10], [5, 1, 20], [5, 20], [3, 4, 15], [10, 50]]
Output
[null, true, true, true, false, false]

Explanation
Bank bank = new Bank([10, 100, 20, 50, 30]);
bank.withdraw(3, 10);    // return true, account 3 has a balance of $20, so it is valid to withdraw $10.
                         // Account 3 has $20 - $10 = $10.
bank.transfer(5, 1, 20); // return true, account 5 has a balance of $30, so it is valid to transfer $20.
                         // Account 5 has $30 - $20 = $10, and account 1 has $10 + $20 = $30.
bank.deposit(5, 20);     // return true, it is valid to deposit $20 to account 5.
                         // Account 5 has $10 + $20 = $30.
bank.transfer(3, 4, 15); // return false, the current balance of account 3 is $10,
                         // so it is invalid to transfer $15 from it.
bank.withdraw(10, 50);   // return false, it is invalid because account 10 does not exist.
```

**Constraints:**

- `n == balance.length`
- `1 <= n, account, account1, account2 <= 10^5`
- `0 <= balance[i], money <= 10^12`
- At most `10^4` calls will be made to **each** function `transfer`, `deposit`, `withdraw`.

JavaScript

```
/**
 * @param {number[]} balance
 */
var Bank = function(balance) {

};

/**
```

```
 * @param {number} account1
 * @param {number} account2
 * @param {number} money
 * @return {boolean}
 */
Bank.prototype.transfer = function(account1, account2, money) {

};

/**
 * @param {number} account
 * @param {number} money
 * @return {boolean}
 */
Bank.prototype.deposit = function(account, money) {

};

/**
 * @param {number} account
 * @param {number} money
 * @return {boolean}
 */
Bank.prototype.withdraw = function(account, money) {

};

/**
 * Your Bank object will be instantiated and called as such:
 * var obj = new Bank(balance)
 * var param_1 = obj.transfer(account1,account2,money)
 * var param_2 = obj.deposit(account,money)
 * var param_3 = obj.withdraw(account,money)
 */
```

☐ **Custom Testcase** Use Example Testcases

Run Submit