BT 3

🎯 PRACTICAL OVERVIEW

A Solidity smart contract named Bank that keeps an internal ledger (balances) for each address and lets users:

- create an account (actually unnecessary, see notes),
- deposit a number into their balance,
- withdraw (subtract) a number from their balance,
- transfer a number from their balance to another address,
- read their balance.

Important: In your current code, **Ether is not actually moved**. The contract does not use msg.value in deposit and does not send ETH back in withdraw. It only updates numbers in a mapping. That's fine for understanding mappings/state, but it's **not a real bank of Ether** yet.

📝 PROBLEM STATEMENT

"Maintain per-user balances and allow deposit, withdraw, transfer, and view balance."

Subject: **BT (Blockchain Technology)**
Topic: **Solidity basics—state variables, mappings, functions, msg.sender, visibility, require, and view.**

🔍 CODE EXPLANATION (every line, every word)

// SPDX-License-Identifier: Bhide License

- // → single-line comment in Solidity.
- SPDX-License-Identifier: → standard header telling tools what license the file uses.
- Bhide License → a custom string you wrote. (Common choices: MIT, GPL-3.0, etc.)

pragma solidity ^0.8.0;

- pragma → compiler instruction.

- solidity → language name.
- ^0.8.0 → compile with version 0.8.0 or higher **up to** (but not including) 0.9.0.

contract Bank {

- contract → starts a new smart contract.
- Bank → the contract's name (used when deploying and calling).

// mapping(type => type)

- Comment reminding the general syntax of a mapping.

mapping(address => uint256) private balances;

- mapping → key→value dictionary stored on-chain.
- (address => uint256) → key type is an Ethereum address; value type is uint256 (unsigned integer).
- private → only code inside **this** contract can access balances directly (other contracts can't).
- balances → the variable name.
  Meaning: balances[addr] stores a number for each address. Default is 0 if never written.

```
function createAccount() public {
    balances[msg.sender] = 0;
}
```

- function createAccount() → defines a function named createAccount.
- public → anyone can call it (and other contracts too).
- msg.sender → the address that called the function (the user/wallet).
- balances[msg.sender] = 0; → sets caller's balance to zero.
  Note: This is redundant: mappings default to 0. Worse, if the caller already had a positive balance, this **resets** it to 0 (danger).

// payable is necessary because the function accepts a value (amount) as a parameter (EXTERNAL SOURCE AHE MHANUN)

```
function deposit(uint256 amount) public payable {
    balances[msg.sender] += amount;
}
```

- Comment: "payable is necessary…" → There's a misunderstanding here:

    - payable means the function **can receive Ether with the transaction** (msg.value).
    - Your function **accepts a numeric parameter** amount, but it **doesn't use** msg.value. So even if the caller sends ETH, you ignore it.

- function deposit(uint256 amount) → takes a number called amount.
- public → callable by anyone.
- payable → allows Ether to be attached, but you didn't read msg.value.
- balances[msg.sender] += amount; → increases the caller's stored number by amount.
  Result: This updates the internal ledger only. It does **not** move real ETH into the contract balance.

```
function withdraw(uint256 amount) public {
   require(balances[msg.sender] >= amount, "Insufficient balance");
   balances[msg.sender] -= amount;
}
```

- function withdraw(uint256 amount) → user wants to subtract amount from their balance.
- public → anyone can call for themselves.
- require(condition, "message") → if condition is false, revert with error message.
- balances[msg.sender] >= amount → must have enough internal balance.
- balances[msg.sender] -= amount; → subtracts from the ledger.
  Missing: It never **sends ETH** back to the user. So wallet funds don't change—only the mapping changes.

```
function transfer(address recipient, uint256 amount) public {
   require(balances[msg.sender] >= amount, "Insufficient balance");
   balances[msg.sender] -= amount;
   balances[recipient] += amount;
}
```

- function transfer(address recipient, uint256 amount) → move internal balance from caller to recipient.
- address recipient → any Ethereum address.

- require(balances[msg.sender] >= amount, ...) → must have enough internal balance.
- Subtract from sender, add to recipient → pure internal ledger transfer.
  Note: This does **not** send ETH to recipient; it only adjusts numbers in the mapping.

```
// view does not modify values within the contract (return kartana lihaycha)
function getBalance() public view returns (uint256) {
    return balances[msg.sender];
}
```

- Comment: view doesn't modify state (correct).
- function getBalance() → returns the caller's stored number.
- public → anyone can call.
- view → read-only, no state change.
- returns (uint256) → returns an unsigned integer.
- return balances[msg.sender]; → reads the mapping for the caller and returns it.

```
}
```

- End of the contract.

⚙️ ALGORITHM & COMPLEXITY

- All operations are single mapping reads/writes → **O(1)** time and space (per operation).
- require checks are constant time.
- No loops, no heavy computation.

But functionally:

- deposit/withdraw/transfer only change numbers in a mapping.
- **No real Ether** flows because msg.value is unused and no Ether is sent out in withdraw.

📥 INPUT REQUIREMENTS (for your current code)

- createAccount() → no inputs (but risky: resets to 0).

- deposit(amount) → input is a plain number (uint256). Even if you send ETH via Remix's "Value" box, **your code ignores it**.
- withdraw(amount) → number to subtract. No ETH is transferred out.
- transfer(recipient, amount) → recipient is an address, amount is a number.
- getBalance() → no input, returns your stored number.

Example calls (Remix VM):

- deposit(100) → your internal balance becomes 100.
- withdraw(40) → balance becomes 60.
- transfer(0xABC..., 10) → your balance −10; recipient's +10.
- getBalance() → returns the number.

📤 OUTPUT EXPLANATION

There's no printed text—only:

- **State changes** in the balances mapping.
- getBalance() returns a number.
- No events were emitted, so you won't see logs unless you add them.

"This contract implements a simple bank using a mapping from address to uint256 to track each user's balance in wei. deposit is payable and credits balances[msg.sender] with msg.value. withdraw verifies sufficient balance, applies Checks-Effects-Interactions, and sends ETH back using call, checking the return flag. getBalance is a view function. I emit Deposit/Withdraw events for observability. I validate inputs and avoid reentrancy by updating state before external calls. I tested on Sepolia via Remix using MetaMask and test ETH."

A) Your Current Code (only updates mapping; no real Ether moves)

Function	Input (how you provide)	Output (what you see)	Side-effects
createAccount()	No input	No return value	Sets balances[msg.sender] = 0 (⚠️ resets any existing balance)
deposit(uint256 amount)	amount (uint) typed in Remix	No return value	Adds amount to balances[msg.sender] (⚠️ ignores any ETH in msg.value)
withdraw(uint256 amount)	amount (uint)	No return value	Subtracts amount from balances[msg.sender] (⚠️ does **not** send ETH)
transfer(address recipient, uint256 amount)	recipient (address), amount (uint)	No return value	Moves amount in the **internal ledger** from caller to recipient (no ETH sent)
getBalance()	No input	**Returns** uint256 (your stored balance)	Read-only; returns the number in mapping

Example run (Remix VM):

- Call deposit(100) → getBalance() returns 100.
- Call withdraw(40) → getBalance() returns 60.
- Call transfer(0xABC…, 10) → your getBalance() returns 50, recipient's internal balance +10.
- No ETH actually moves in or out of wallets.

B) Correct Ether Bank (real deposit/withdraw using msg.value)

Function	Input (how you provide)	Output (what you see)	Side-effects
deposit()	**Set Remix "Value"**: e.g., 0.1 ether	No return value; **Deposit event**	Credits balances[msg.sender] += msg.value and contract **receives ETH**

Function	Input (how you provide)	Output (what you see)	Side-effects
withdraw(uint256 amount)	amount in **wei** (e.g., 100000000000000000 for 0.1 ETH)	No return value; **Withdraw event**	Deducts from ledger, **sends ETH** back to your wallet
getBalance()	No input	**Returns** uint256 (wei)	Read-only; shows your on-contract balance

Example run (Remix + MetaMask on Sepolia):

1. Set **Value = 0.05 ether** → call deposit() → getBalance() returns 50000000000000000.
2. Call withdraw(20000000000000000) (0.02 ETH) → wallet receives 0.02 ETH; getBalance() returns 30000000000000000.

Quick Notes You Can Say in Viva

- **Inputs** are either function parameters (e.g., amount, recipient) or **ETH value** attached to the transaction (msg.value) for payable functions like deposit().
- **Outputs** in smart contracts are usually:
    1. **Return values** (e.g., getBalance() returns a number in wei),
    2. **State changes** (mapping updates),
    3. **Events** (logs visible in Remix/Etherscan),
    4. **ETH transfers** (in corrected version's withdraw).

Your Code (with line numbers)

```
1  // SPDX-License-Identifier: Bhide License
2  pragma solidity ^0.8.0;
3
4  contract Bank {
5      // mapping(type => type)
6      mapping(address => uint256) private balances;
7
8      function createAccount() public {
9          balances[msg.sender] = 0;
10     }
11
12     // payable is necessary because the function accepts a value (amount) as a
parameter (EXTERNAL SOURCE AHE MHANUN)
13     function deposit(uint256 amount) public payable {
14         balances[msg.sender] += amount;
15     }
16
17     function withdraw(uint256 amount) public {
18         require(balances[msg.sender] >= amount, "Insufficient balance");
19         balances[msg.sender] -= amount;
20     }
21
22     function transfer(address recipient, uint256 amount) public {
23         require(balances[msg.sender] >= amount, "Insufficient balance");
24         balances[msg.sender] -= amount;
25         balances[recipient] += amount;
26     }
27
28    // view does not modify values within the contract (return kartana lihaycha)
29    function getBalance() public view returns (uint256) {
30        return balances[msg.sender];
31    }
32 }
```

Line-by-line explanation

1 // SPDX-License-Identifier: Bhide License

- // starts a single-line comment.
- SPDX-License-Identifier is a standard header for license tooling.
- "Bhide License" is just a string you wrote; typical values are MIT, GPL-3.0, etc.

2 pragma solidity ^0.8.0;

- Compiler directive: compile with Solidity version **0.8.0 or newer** (but <0.9.0).
- Solidity 0.8+ has built-in overflow/underflow checks.

3 (blank)

- Just spacing for readability.

4 contract Bank {

- Begins a new **smart contract** named **Bank**.
- Everything between { ... } is the contract's code and state.

5 // mapping(type => type)

- Comment reminding mapping syntax.

6 mapping(address => uint256) private balances;

- Declares a **state variable** named balances.
- Type: a **mapping** from address → uint256.
    - Key: an Ethereum address (EOA or contract).
    - Value: an unsigned 256-bit integer.
- private: only functions **inside this contract** can access balances directly.

- Default for any balances[someAddress] is **0** until written.

7 (blank)

8 function createAccount() public {

- Declares a **public** function createAccount() (anyone can call).
- No inputs, no outputs.

9 balances[msg.sender] = 0;

- msg.sender is **the caller's address**.
- Sets their stored balance to **0** explicitly.
  - ⚠️ **Note:** Mappings already default to 0. This line is **redundant** if the user is new.
  - ⚠️ If the user had a **non-zero** balance, this **resets it to 0** (dangerous).

10 }

- Ends createAccount.

11 (blank)

12 // payable is necessary because the function accepts a value (amount) as a parameter (EXTERNAL SOURCE AHE MHANUN)

- Comment: mixes ideas. In Solidity, **payable** means the function can **receive Ether** (msg.value).
- It is **not** about "accepting a numeric parameter"; any function can accept a number.

13 function deposit(uint256 amount) public payable {

- Function deposit takes a **number** amount and is **payable** (can receive ETH).
- public: callable by anyone.
- payable: allows sending Ether with the transaction (in Remix "Value" field).
  - ⚠️ **But your function never uses msg.value.** So any Ether sent is **ignored** in logic.

14 balances[msg.sender] += amount;

- Increases the caller's **internal ledger** by the *parameter* amount.
- ⚠️ **No real Ether is credited** here; just a number in the mapping changes.

- If the caller set Remix “Value” to some ETH, that ETH sits in the contract (or tx fails depending on context) but is **not** tied to this balance.

15 }

- Ends deposit.

16 (blank)

17 function withdraw(uint256 amount) public {

- Public function to withdraw a **number** from your internal balance.
- No payable needed (not receiving ETH).

18 require(balances[msg.sender] >= amount, "Insufficient balance");

- Guard check: if caller’s internal balance < amount, revert with message.
- require reverts the whole transaction on failure (state changes undone).

19 balances[msg.sender] -= amount;

- Subtracts amount from the internal ledger.
- ⚠️ **Does not send Ether** to the caller’s wallet. Only the mapping value changes.

20 }

- Ends withdraw.

21 (blank)

22 function transfer(address recipient, uint256 amount) public {

- Public function to move internal balance from caller to **recipient**.
- Input 1: recipient is an **Ethereum address**.
- Input 2: amount is a number.

23 require(balances[msg.sender] >= amount, "Insufficient balance");

- Check: caller must have at least amount in the ledger.

24 balances[msg.sender] -= amount;

- Deducts from caller’s internal balance.

25 balances[recipient] += amount;

- Adds to recipient’s internal balance.
- ⚠️ **No Ether is sent** to the recipient address; it’s just the mapping.

26 }

- Ends transfer.

27 (blank)

28 // view does not modify values within the contract (return kartana lihaycha)

- Comment: view means **read-only** (cannot modify state).

29 function getBalance() public view returns (uint256) {

- Public **view** function that **returns** a uint256 (the caller's balance).

30 return balances[msg.sender];

- Reads the mapping for the caller and returns the number (default 0 if never set).

31 }

- Ends getBalance.

32 }

- Ends the Bank contract.