Practial 3: Smart contract creation in channel

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

/// @title SimpleChannel - a lightweight on-chain "channel" for posts + tipping
/// @notice Demonstrates storage, events, access control, payable functions, and view helpers
contract SimpleChannel {
    address public owner;
    uint256 public postCount;

    struct Post {
        uint256 id;
        address author;
        string content;
        uint256 timestamp;
        uint256 tipsWei;
        bool deleted;
    }

    // postId -> Post
    mapping(uint256 => Post) private posts;

    // events
    event PostCreated(uint256 indexed id, address indexed author, string content, uint256 timestamp);
    event PostTipped(uint256 indexed id, address indexed tipper, uint256 amount);
    event PostDeleted(uint256 indexed id, address indexed deletedBy);

    modifier onlyOwner() {
        require(msg.sender == owner, "Only owner");
```

```
    _;
}

modifier exists(uint256 _id) {
    require(_id > 0 && _id <= postCount, "Post does not exist");
    require(!posts[_id].deleted, "Post deleted");
    _;
}

constructor() {
    owner = msg.sender;
    postCount = 0;
}

/// @notice Create a new post in the channel
/// @param _content The text content of the post
function createPost(string calldata _content) external {
    require(bytes(_content).length > 0, "Content empty");

    postCount += 1;
    posts[postCount] = Post({
        id: postCount,
        author: msg.sender,
        content: _content,
        timestamp: block.timestamp,
        tipsWei: 0,
        deleted: false
    });

    emit PostCreated(postCount, msg.sender, _content, block.timestamp);
}
```

```
/// @notice Tip a post (send ETH to the author)
/// @param _id The post id
function tipPost(uint256 _id) external payable exists(_id) {
    require(msg.value > 0, "Tip must be > 0");
    Post storage p = posts[_id];
    p.tipsWei += msg.value;

    // forward tip to author
    (bool sent, ) = p.author.call{value: msg.value}("");
    require(sent, "Transfer failed");

    emit PostTipped(_id, msg.sender, msg.value);
}

/// @notice Mark a post deleted (only owner can permanently delete)
/// @param _id The post id
function deletePost(uint256 _id) external onlyOwner exists(_id) {
    posts[_id].deleted = true;
    emit PostDeleted(_id, msg.sender);
}

/// @notice Get details of a post
/// @param _id The post id
function getPost(uint256 _id) external view returns (
    uint256 id,
    address author,
    string memory content,
    uint256 timestamp,
    uint256 tipsWei,
    bool deleted
```

```
    ) {
        Post storage p = posts[_id];
        return (p.id, p.author, p.content, p.timestamp, p.tipsWei, p.deleted);
    }

    /// @notice Get multiple posts (range) - useful for simple frontends
    /// @param _from starting id (inclusive)
    /// @param _to ending id (inclusive)
    function getPostsRange(uint256 _from, uint256 _to) external view returns (Post[] memory) {
        if (_from < 1) _from = 1;
        if (_to > postCount) _to = postCount;
        require(_from <= _to, "Invalid range");

        uint256 len = _to - _from + 1;
        Post[] memory arr = new Post[](len);
        uint256 idx = 0;
        for (uint256 i = _from; i <= _to; ++i) {
            arr[idx] = posts[i];
            idx++;
        }
        return arr;
    }

    // allow the contract to receive ETH (not used but good to have)
    receive() external payable {}
    fallback() external payable {}
}
```