Solidity Events Quiz

Quiz 1: Understanding Events

Instructions: Neri is creating a transparent donation system for the barangay. What is the main purpose of using events in Solidity?

```
pragma solidity ^0.8.0;

contract BarangayFund {
    uint256 public totalDonations;

    // Event declaration
    event DonationReceived(address donor, uint256 amount, string message);

    function donate(string memory message) public payable {
        totalDonations += msg.value;

        // Event emission
        emit DonationReceived(msg.sender, msg.value, message);
    }
}
```

What is the main purpose of the DonationReceived event in this contract?

- A) It automatically sends money to the donor
- B) It logs donation information that external applications can track
- C) It prevents duplicate donations
- D) It updates the contract's balance

Answer: B) It logs donation information that external applications can track

Explanation: Events in Solidity are like announcements or notifications that a contract makes when something important happens. In this example, whenever someone donates, the contract emits (broadcasts) a `DonationReceived` event with details about who donated, how much, and what message they included.

These events are stored in the blockchain's transaction logs (not in the contract's storage), making them:

1. Cheaper than storing all this history in contract variables
2. Easily searchable by applications outside the blockchain
3. Perfect for creating transparent records that can't be changed

For example, a barangay website could listen for these events to show a live donation feed or create reports of all donations received.

Quiz 2: Event Parameters

Instructions: Neri wants to make it easy to search for specific donations in her system. Which of the following correctly explains the `indexed` keyword in events?

```
pragma solidity ^0.8.0;

contract EnhancedDonations {
    event FundTransfer(
        address indexed from,
        address indexed to,
        uint256 amount,
        string message
    );

    function transferFunds(address recipient, string memory message) public
payable {
        // Transfer logic here

        emit FundTransfer(msg.sender, recipient, msg.value, message);
    }
}
```

What does the `indexed` keyword do in the event declaration?

- A) It makes those parameters more important than others
- B) It allows searching/filtering for specific values of those parameters
- C) It encrypts those parameters for privacy
- D) It makes those parameters visible on the blockchain explorer

Answer: B) It allows searching/filtering for specific values of those parameters

Explanation: The `indexed` keyword creates something like a searchable index for that parameter. When a parameter is marked as `indexed`, its value is stored in a special way that makes it much easier and faster to search for specific values.

For example, with this event, you could easily search for:

- All fund transfers from a particular address
- All fund transfers to a specific recipient

This is especially useful for building user interfaces that need to filter through many events. Think of it like adding a hashtag (#) to social media posts so they can be found more easily. Without the `indexed` keyword, you'd have to download and check every single event to find what you're looking for.

Note: You can have up to three indexed parameters per event.

Quiz 3: Emitting Events

Instructions: Neri is updating her barangay fund tracker. Which option correctly emits an event when funds are withdrawn?

```
pragma solidity ^0.8.0;

contract BarangayFundTracker {
    address public barangayOfficial;
    uint256 public totalFunds;

    event FundWithdrawal(address official, uint256 amount, uint256 timestamp);

    constructor() {
        barangayOfficial = msg.sender;
    }

    function depositFunds() public payable {
        totalFunds += msg.value;
    }

    // Option A
    function withdrawFunds1(uint256 amount) public {
        require(msg.sender == barangayOfficial, "Not authorized");
        require(amount <= totalFunds, "Insufficient funds");
        totalFunds -= amount;
        payable(barangayOfficial).transfer(amount);
        FundWithdrawal(barangayOfficial, amount, block.timestamp);
    }

    // Option B
    function withdrawFunds2(uint256 amount) public {
        require(msg.sender == barangayOfficial, "Not authorized");
        require(amount <= totalFunds, "Insufficient funds");
        totalFunds -= amount;
        payable(barangayOfficial).transfer(amount);
        emit FundWithdrawal(barangayOfficial, amount, block.timestamp);
    }

    // Option C
    function withdrawFunds3(uint256 amount) public {
        require(msg.sender == barangayOfficial, "Not authorized");
        require(amount <= totalFunds, "Insufficient funds");
        FundWithdrawal(barangayOfficial, amount, block.timestamp);
        totalFunds -= amount;
        payable(barangayOfficial).transfer(amount);
    }

    // Option D
    function withdrawFunds4(uint256 amount) public {
        require(msg.sender == barangayOfficial, "Not authorized");
        require(amount <= totalFunds, "Insufficient funds");
        totalFunds -= amount;
        payable(barangayOfficial).transfer(amount);
    }
}
```

Which function correctly emits the event when funds are withdrawn?

- A) Option A
- B) Option B
- C) Option C
- D) Option D

Answer: B) Option B

Explanation: Option B correctly emits the event because:

1. It uses the required `emit` keyword before the event name
2. It emits the event after the state changes have been made (withdrawing funds)
3. It passes all the required parameters in the right order

Option A is missing the `emit` keyword, which has been required since Solidity 0.4.21. Option C emits the event before making the state changes, which is bad practice (events should report what has happened, not what will happen). Option D doesn't emit the event at all, which means there would be no record of withdrawals for external applications to track.

Quiz 4: Multiple Events

Instructions: Neri wants her contract to log different types of activities. What's the best practice for naming events in a contract with multiple events?

```
pragma solidity ^0.8.0;

contract MultiEventTracker {
    // Option A
    event e1(address user, uint256 amount);
    event e2(address user, string action);

    // Option B
    event UserDeposit(address user, uint256 amount);
    event UserAction(address user, string action);

    // Option C
    event DepositMade(address user, uint256 amount);
    event ActionPerformed(address user, string action);

    // Option D
    event DEPOSIT_EVENT(address user, uint256 amount);
    event ACTION_EVENT(address user, string action);
}
```

Which naming convention is best for events?

- A) Option A
- B) Option B
- C) Option C

- D) Option D

Answer: C) Option C

Explanation: Option C follows the best practices for naming events because:

1. The names are descriptive and clearly indicate what happened (`DepositMade`, `ActionPerformed`)
2. They use "past tense" verb phrases, indicating that something has already occurred
3. They follow standard camel case naming convention (starting with a capital letter)

This makes the code much more readable and self-documenting. When someone looks at code that emits `DepositMade`, they immediately understand what that event represents.

Option A uses meaningless names (e1, e2) that don't describe what the events represent. Option B uses names that describe the object but not the action (what happened to the user?). Option D uses ALL_CAPS which is typically reserved for constants in Solidity, not events.