Different Contracts and the functionalities:-

Blockchain Contract:

- **addBlock()**: Adds a new block with given insurance contracts and the previous hash. Calculates the hash of the new block and adds it to the chain.
- **calculateHash()**: Computes the hash of a block based on its properties.
- **validateChain()**: Checks the integrity of the blockchain by verifying the hash of each block and its link to the previous block.

InsuranceContract Contract:

- **signContract()**: Activates the contract.
- **isValid()**: Checks whether the contract is valid based on its terms.
- **addTerm()**: Adds a new term to the insurance contract.

PolicyHolder Contract:

- **applyForInsurance()**: Adds a new insurance contract to the policyholder's list of policies.

Insurer Contract:

- **offerInsurance()**: Adds a new insurance contract to the insurer's list of offered policies.

Term Contract:

- **addCondition()**: Adds a condition to the term.
- **evaluate()**: Checks if all conditions in the term are satisfied.

Condition Contract:

- **submitClaim()**: Sets the condition status to true (claim submitted).
- **updateStatus()**: Updates the status of the condition.
- **status()**: Returns the current status of the condition.

SmartContract Contract:

- **executePayment()**: Transfers the specified amount to the recipient if the contract is active.
- **validatePolicy()**: Validates if the given insurance contract is valid.
- **fileClaim()**: Submits a claim for a given condition.

Transaction Contract:

- Represents a transaction with fromAddress, toAddress, and amount properties.

Functionalities of each contract

1. PolicyHolder Contract

This contract represents the entity holding the policy. It has the following buttons:

- **applyForInsurance()**
  - **Description**: This function allows the policyholder to apply for insurance.
  - **Input**: None (assuming the PolicyHolder contract already holds information about the policyholder).
  - **Execution**: Click this button when the policyholder wants to apply for a new insurance contract.
  - **Output**: The policyholder's application is submitted.
- **holderId()**
  - **Description**: Returns the unique identifier for the policyholder.
  - **Input**: None (read-only).
  - **Output**: Returns a string representing the policyholder's ID.
- **policies()**
  - **Description**: This function returns the list of policies associated with the policyholder.
  - **Input**: None (read-only).
  - **Output**: Array of insurance contract addresses.

2. Insurer Contract

This contract represents the insurance company. It has the following buttons:

- **offerInsurance()**
  - **Description**: This function allows the insurer to offer a new insurance policy to a policyholder.
  - **Input**: None (assuming the insurer's details are known).
  - **Execution**: Click this button to offer a new insurance contract.
  - **Output**: A new insurance offer is made.
- **insuranceId()**
  - **Description**: Returns the unique identifier for the insurer.
  - **Input**: None (read-only).
  - **Output**: Returns a string representing the insurer's ID.
- **offeredPolicies()**
  - **Description**: This function returns the list of policies offered by the insurer.
  - **Input**: None (read-only).
  - **Output**: Array of offered insurance contract addresses.

3. Term Contract

This contract represents the terms of an insurance policy. It has the following buttons:

- **addCondition(address _condition)**
  - **Description**: Adds a new condition to the term.
  - **Input**:
    - _condition: The address of a Condition contract.
  - **Execution**: Click this button after deploying a Condition contract.
  - **Output**: The condition is added to the term.
- **conditions()**
  - **Description**: Returns the list of conditions associated with this term.
  - **Input**: None (read-only).
  - **Output**: Array of condition contract addresses.
- **evaluate()**
  - **Description**: Evaluates all conditions in the term.
  - **Input**: None.
  - **Execution**: Click to check if all conditions in this term are met.
  - **Output**: Boolean indicating whether all conditions are satisfied.
- **termId()**
  - **Description**: Returns the unique identifier for the term.
  - **Input**: None (read-only).
  - **Output**: Returns a string representing the term ID.
- **termType()**
  - **Description**: Returns the type of the term (e.g., "Health", "Life").
  - **Input**: None (read-only).
  - **Output**: Returns a string representing the term type.

4. Condition Contract

This contract represents a specific condition within a term. It has the following buttons:

- **submitClaim()**
  - **Description**: Submits a claim to be evaluated against the condition.
  - **Input**: None.
  - **Execution**: Click this button to submit a claim.
  - **Output**: The claim is submitted and the status might change based on evaluation.

- **updateStatus(uint _status)**
    - **Description**: Updates the status of the condition manually.
    - **Input**:
        - _status: A number representing the new status of the condition.
    - **Execution**: Click to update the status.
    - **Output**: The status is updated.
- **claimId()**
    - **Description**: Returns the claim identifier.
    - **Input**: None (read-only).
    - **Output**: Returns an integer representing the claim ID.
- **policyNumber()**
    - **Description**: Returns the policy number associated with this condition.
    - **Input**: None (read-only).
    - **Output**: Returns a string representing the policy number.
- **status()**
    - **Description**: Returns the status of the condition.
    - **Input**: None (read-only).
    - **Output**: Returns the current status of the condition (e.g., 0 for pending, 1 for approved).

5. InsuranceContract

This contract represents the insurance agreement. It has the following buttons:

- **addTerm(address _term)**
    - **Description**: Adds a new term to the insurance contract.
    - **Input**:
        - _term: The address of a Term contract.
    - **Execution**: Click to add a term to the insurance contract.
    - **Output**: The term is added to the contract.
- **signContract()**
    - **Description**: Signs the insurance contract, making it active.
    - **Input**: None.
    - **Execution**: Click to activate the contract after all terms and conditions have been added.
    - **Output**: The contract is now active and enforceable.
- **contractId()**

    - **Description**: Returns the unique identifier for the insurance contract.
    - **Input**: None (read-only).
    - **Output**: Returns a string representing the contract ID.
- **insurer()**
    - **Description**: Returns the address of the insurer.
    - **Input**: None (read-only).
    - **Output**: Returns the address of the insurer.
- **isActive()**
    - **Description**: Indicates if the contract is active.
    - **Input**: None (read-only).
    - **Output**: Returns a boolean indicating whether the contract is active.
- **isValid()**
    - **Description**: Indicates if the contract is valid based on its terms and conditions.
    - **Input**: None (read-only).
    - **Output**: Returns a boolean indicating whether the contract is valid.
- **policyholder()**
    - **Description**: Returns the address of the policyholder.
    - **Input**: None (read-only).
    - **Output**: Returns the address of the policyholder.
- **terms()**
    - **Description**: Returns the list of terms associated with this insurance contract.
    - **Input**: None (read-only).
    - **Output**: Array of term contract addresses.

6. SmartContract

This contract manages the execution of the insurance contract. It has the following buttons:

- **executePayment()**
    - **Description**: Executes payment based on predefined conditions and terms.
    - **Input**: None.
    - **Execution**: Click to execute a payment.
    - **Output**: Payment is made according to contract terms.
- **fileClaim()**
    - **Description**: Files a claim with the smart contract.

    - **Input**: None.
    - **Execution**: Click to file a claim against the insurance contract.
    - **Output**: Claim is filed and evaluated.
- **contractAddress()**
    - **Description**: Returns the address of the associated insurance contract.
    - **Input**: None (read-only).
    - **Output**: Address of the associated insurance contract.
- **isActive()**
    - **Description**: Indicates if the smart contract is active.
    - **Input**: None (read-only).
    - **Output**: Returns a boolean indicating whether the smart contract is active.
- **owner()**
    - **Description**: Returns the owner of the smart contract.
    - **Input**: None (read-only).
    - **Output**: Address of the smart contract owner.
- **validatePolicy()**
    - **Description**: Validates the policy by checking all terms and conditions.
    - **Input**: None.
    - **Execution**: Click to validate the insurance policy.
    - **Output**: Boolean indicating whether the policy is valid.

Step-by-Step Guide

1. Deploy PolicyHolder

- **Select PolicyHolder from the dropdown menu**.
- **Constructor Parameters**:
  - string _holderId: A unique identifier for the policyholder, e.g., "PH1".
- **Deploy**:
  - Click the Deploy button.
  - Copy the deployed contract address for use in the InsuranceContract.

2. Deploy Insurer

- **Select Insurer from the dropdown menu**.
- **Constructor Parameters**:
  - string _insurerId: A unique identifier for the insurer, e.g., "INS1".
- **Deploy**:
  - Click the Deploy button.
  - Copy the deployed contract address for use in the InsuranceContract.

3. Deploy Term

- **Select Term from the dropdown menu**.
- **Constructor Parameters**:
  - string _termId: A unique identifier for the term, e.g., "T1".
  - string _type: The type of the term, e.g., "Health".
  - address[] _conditions: Array of Condition contract addresses (can be empty initially).
- **Deploy**:
  - Click the Deploy button.
  - Copy the deployed contract address for use in the InsuranceContract.

4. Deploy Condition (if needed)

- **Select Condition from the dropdown menu**.
- **Constructor Parameters**:
  - int _claimId: An integer claim identifier, e.g., 1.
  - string _policyNumber: Policy number associated with the condition, e.g., "POL123".
- **Deploy**:
  - Click the Deploy button.
  - Copy the deployed contract address for use in the Term.

5. Deploy InsuranceContract

- **Select InsuranceContract from the dropdown menu**.
- **Constructor Parameters**:
  - string _contractID: A unique identifier for the insurance contract, e.g., "IC001".
  - address _policyHolder: Address of the deployed PolicyHolder contract.
  - address _insurer: Address of the deployed Insurer contract.
  - address[] _terms: Array of Term contract addresses.
- **Deploy**:
  - Click the Deploy button.
  - Copy the deployed contract address for use in the Blockchain.

6. Deploy SmartContract (if needed)

- **Select SmartContract from the dropdown menu**.
- **Constructor Parameters**:
  - string _contractAddress: The address of the deployed InsuranceContract.
  - string _owner: Owner of the smart contract (your wallet address).
  - bool _isActive: Set to true if the contract is active.
- **Deploy**:
  - Click the Deploy button.
  - This contract will be used for executing and validating transactions based on predefined conditions.

7. Deploy Blockchain

- **Select Blockchain from the dropdown menu**.
- **Constructor Parameters**: None.
- **Deploy**:
  - Click the Deploy button.

8. Add Condition to Term

- **Action**: Go to the deployed Term contract.
- **Function**: addCondition(address _condition).
- **Input**:
  - **_condition**: The address of the deployed Condition contract.
- **Execution**: Click Add Condition.
- **Output**: The condition is now linked to the term. You can verify by calling the conditions() function to see the updated list of conditions associated with this term.

9. **Add Term to InsuranceContract**

- **Action**: Go to InsuranceContract.
- **Function**: addTerm(address _term).
- **Input**: Address of the deployed Term.
- **Execution**: Click Add Term.
- **Output**: The term is now linked to the insurance contract

10. Add InsuranceContract to the Blockchain

- **Select the deployed Blockchain instance**.
- **Method addBlock**:
    - **Parameters**:
        - address[] contracts: An array of deployed InsuranceContract addresses.
        - bytes32 previousHash: The hash of the previous block (use "0x0" for the first block).
    - **Operation**:
        - Add one or more deployed InsuranceContract instances to a new block.

11. Validate the Blockchain

- **Select the validateChain function**.
- **Operation**:
    - Call validateChain() to check the integrity of the blockchain.

12. Sign the Insurance Contract

- **Go to the InsuranceContract Contract**
    - Use the signContract() button.
    - **Description**: This function activates the contract. Make sure this step is executed by the policyholder or the insurer as per the contract design.
    - **Input**: None.
    - **Expected Outcome**: The contract should now be marked as active. You can check this by calling the isActive() function, which should return true.

13. Policyholder Applies for Insurance

- **Go to the PolicyHolder Contract**
    - Use the applyForInsurance() button.
    - **Description**: The policyholder uses this function to apply for the insurance that has been offered by the insurer.
    - **Input**: None (ensure the policyholder is associated with the correct insurance contract).

    - **Expected Outcome**: The policyholder's application should be processed, and their policy should now be linked to the InsuranceContract.

14. Simulate a Claim Condition

- If a condition is related to a specific event (e.g., an accident or an expiration date), simulate that condition.
- **Go to the Condition Contract**
    - Use the submitClaim() button.
    - **Description**: This function should simulate the event, like the occurrence of an accident.
    - **Input**: None.
    - **Expected Outcome**: The claim associated with this condition should now be pending evaluation. Check the condition's status() for updates.

15. Evaluate the Condition

- **Go to the Term Contract**
    - Use the evaluate() button.
    - **Description**: This function evaluates all the conditions within the term. If all conditions are met, the term is considered valid.
    - **Input**: None.
    - **Expected Outcome**: The term should now be validated. You can check whether the term has been fulfilled by calling any relevant getter function, like conditions(), to see their statuses.

16. File a Claim on the SmartContract

- **Go to the SmartContract Contract**
    - Use the fileClaim() button.
    - **Description**: The policyholder calls this function to file a claim after the conditions have been evaluated.
    - **Input**: None.
    - **Expected Outcome**: The claim will be processed based on the results of the evaluated conditions. The status of the claim should now reflect whether it is approved or denied.

17. Execute Payment

- **Go to the SmartContract Contract**
    - Use the executePayment() button.
    - **Description**: If the claim has been approved, this function will execute the payment to the policyholder based on the insurance contract.
    - **Input**: None.
    - **Expected Outcome**: The payment should be made to the policyholder as per the contract terms. You can check the contract's balance or the policyholder's balance to confirm this.

18. Check Contract States

- After executing the payment, you should check the state of the involved contracts:
  - **InsuranceContract**
    - Use isValid() and isActive() to check if the contract is still valid and active.
    - Check policyholder() and insurer() to confirm that they are still linked to the contract.
  - **SmartContract**
    - Use isActive() to confirm that the contract is still active.
    - Use validatePolicy() to see if the policy remains valid.