// SPDX-License-Identifier: MIT

pragma solidity ^0.8.18;

// Contracts 1–15 are above. Below are contracts 16–20:

/// ------------------------------

/// 16. Smart Contract Update Proxy System (UUPS-like)

/// ------------------------------

contract LogicV1 {

uint public value;

function setValue(uint \_value) public {

value = \_value;

}

}

contract Proxy {

address public implementation;

address public admin;

constructor(address \_implementation) {

implementation = \_implementation;

admin = msg.sender;

}

function upgrade(address \_newImpl) public {

require(msg.sender == admin, "Not admin");

implementation = \_newImpl;

}

fallback() external payable {

address impl = implementation;

require(impl != address(0));

assembly {

calldatacopy(0, 0, calldatasize())

let result := delegatecall(gas(), impl, 0, calldatasize(), 0, 0)

returndatacopy(0, 0, returndatasize())

switch result

case 0 { revert(0, returndatasize()) }

default { return(0, returndatasize()) }

}

}

}

/// ------------------------------

/// 17. Cross-Chain NFT Bridge (Mocked)

/// ------------------------------

contract NFTBridge {

mapping(uint => string) public tokensOnHold;

function depositNFT(uint tokenId, string memory chain) public {

tokensOnHold[tokenId] = chain;

}

function releaseNFT(uint tokenId) public {

require(bytes(tokensOnHold[tokenId]).length != 0, "Not bridged");

delete tokensOnHold[tokenId];

}

}

/// ------------------------------

/// 18. Decentralized Insurance Pool

/// ------------------------------

contract InsurancePool {

address public admin;

mapping(address => uint) public stakes;

uint public totalStaked;

struct Claim {

address claimant;

uint amount;

bool approved;

}

Claim[] public claims;

constructor() {

admin = msg.sender;

}

function stake() external payable {

stakes[msg.sender] += msg.value;

totalStaked += msg.value;

}

function submitClaim(uint amount) public {

claims.push(Claim(msg.sender, amount, false));

}

function approveClaim(uint index) public {

require(msg.sender == admin);

Claim storage c = claims[index];

require(!c.approved);

c.approved = true;

payable(c.claimant).transfer(c.amount);

}

receive() external payable {}

}

/// ------------------------------

/// 19. CI/CD-integrated Bug Bounty Contract

/// ------------------------------

contract BugBounty {

address public owner;

mapping(address => uint) public rewards;

constructor() payable {

owner = msg.sender;

}

function submitReport(address reporter, string memory issueHash) public {

require(msg.sender == owner);

rewards[reporter] += 1 ether;

}

function claimReward() public {

uint reward = rewards[msg.sender];

require(reward > 0);

rewards[msg.sender] = 0;

payable(msg.sender).transfer(reward);

}

receive() external payable {}

}

/// ------------------------------

/// 20. Self-Healing Deployment Contract (Mocked)

/// ------------------------------

contract SelfHealingContract {

address public owner;

address public stableVersion;

address public betaVersion;

bool public degraded;

constructor(address \_stable, address \_beta) {

owner = msg.sender;

stableVersion = \_stable;

betaVersion = \_beta;

degraded = false;

}

function reportDegradation() external {

// Could be triggered by off-chain monitoring bot

degraded = true;

}

function revertToStable() public {

require(degraded, "No degradation reported");

betaVersion = stableVersion;

degraded = false;

}

function currentImplementation() external view returns (address) {

return betaVersion;

}

}