```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.0 <0.8.0;

library SafeMath {

  function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
    uint256 c = a + b;
    if (c < a) return (false, 0);
    return (true, c);
  }

  function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
    if (b > a) return (false, 0);
    return (true, a - b);
  }

  function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
    if (a == 0) return (true, 0);
    uint256 c = a * b;
    if (c / a != b) return (false, 0);
    return (true, c);
  }

  function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
    if (b == 0) return (false, 0);
    return (true, a / b);
  }

  function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
    if (b == 0) return (false, 0);
    return (true, a % b);
```

```
}

function add(uint256 a, uint256 b) internal pure returns (uint256) {
  uint256 c = a + b;
  require(c >= a, "SafeMath: addition overflow");
  return c;
}

function sub(uint256 a, uint256 b) internal pure returns (uint256) {
  require(b <= a, "SafeMath: subtraction overflow");
  return a - b;
}

function mul(uint256 a, uint256 b) internal pure returns (uint256) {
  if (a == 0) return 0;
  uint256 c = a * b;
  require(c / a == b, "SafeMath: multiplication overflow");
  return c;
}

function div(uint256 a, uint256 b) internal pure returns (uint256) {
  require(b > 0, "SafeMath: division by zero");
  return a / b;
}

function mod(uint256 a, uint256 b) internal pure returns (uint256) {
  require(b > 0, "SafeMath: modulo by zero");
  return a % b;
}

function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
    require(b <= a, errorMessage);
    return a - b;
  }

  function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
    require(b > 0, errorMessage);
    return a / b;
  }

  function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
    require(b > 0, errorMessage);
    return a % b;
  }
}

interface IERC20 {

  function totalSupply() external view returns (uint256);

  function balanceOf(address account) external view returns (uint256);

  function transfer(address recipient, uint256 amount) external returns (bool);

  function allowance(address owner, address spender) external view returns (uint256);

  function approve(address spender, uint256 amount) external returns (bool);

  function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

  event Transfer(address indexed from, address indexed to, uint256 value);
  event Approval(address indexed owner, address indexed spender, uint256 value);
}

abstract contract Context {
  function _msgSender() internal view returns (address payable) {
    return msg.sender;
  }

  function _msgData() internal view returns (bytes memory) {
    this; // silence state mutability warning without generating bytecode - see
https://github.com/ethereum/solidity/issues/2691
    return msg.data;
  }
}

abstract contract Ownable is Context {
  address private _owner;
  address private _newOwner;

  event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

  constructor () {
    address msgSender = 0x93830567238254E178fc39A093b078CFf3427d81;
    _owner = msgSender;
    emit OwnershipTransferred(address(0), msgSender);
  }

  function owner() public view returns (address) {
    return _owner;
  }
  modifier onlyOwner() {
    require(owner() == _msgSender(), "Ownable: caller is not the owner");
    _;
  }

  function acceptOwnership() public {
    require(_msgSender() == _newOwner, "Ownable: only new owner can accept ownership");
    address oldOwner = _owner;
    _owner = _newOwner;
    _newOwner = address(0);
    emit OwnershipTransferred(oldOwner, _owner);
  }

  function transferOwnership(address newOwner) public onlyOwner {
    require(newOwner != address(0), "Ownable: new owner is the zero address");
    _newOwner = newOwner;
  }
}

contract AccentCoin is Context, Ownable, IERC20 {
  using SafeMath for uint256;

  mapping (address => uint256) private _balances;
  mapping (address => mapping (address => uint256)) private _allowances;



  uint256 private _totalSupply;

  string private _name;
  string private _symbol;
  uint8 private _decimals;
```

```
constructor() {
  uint256 fractions = 10 ** uint256(18);
  _name = "Accent Coin";
  _symbol = "AC";
  _decimals = 18;
  _totalSupply = 500000000000 * fractions;



  _balances[owner()] = _totalSupply;
  emit Transfer(address(0), owner(), _totalSupply);
}



function name() public view returns (string memory) {
  return _name;
}

function symbol() public view returns (string memory) {
  return _symbol;
}

function decimals() public view returns (uint8) {
  return _decimals;
}

function totalSupply() public view override returns (uint256) {
  return _totalSupply;
}

function balanceOf(address account) public view override returns (uint256) {
    return _balances[account];
  }


  function transfer(address recipient, uint256 amount) public override returns (bool) {
    _transfer(_msgSender(), recipient, amount);
    return true;
  }

  function allowance(address owner, address spender) public view override returns (uint256) {
    return _allowances[owner][spender];
  }

  function approve(address spender, uint256 amount) public override returns (bool) {
    _approve(_msgSender(), spender, amount);
    return true;
  }

  function transferFrom(address sender, address recipient, uint256 amount) public override returns
(bool) {
    _transfer(sender, recipient, amount);
    _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20:
transfer amount exceeds allowance"));
    return true;
  }

  function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
    _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
    return true;
  }

  function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
    _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue,
"ERC20: decreased allowance below zero"));

    return true;

  }


  function mint(address account, uint256 amount) public onlyOwner {

    _mint(account, amount);

    }




      function burn(uint256 amount) public {

        _burn(_msgSender(), amount);

      }



function burnFrom(address account, uint256 amount) public {

    uint256 decreasedAllowance = allowance(account, _msgSender()).sub(amount, "ERC20: burn
amount exceeds allowance");


    _approve(account, _msgSender(), decreasedAllowance);

    _burn(account, amount);

  }




  function _transfer(address sender, address recipient, uint256 amount) internal {

    require(sender != address(0), "ERC20: transfer from the zero address");

    require(recipient != address(0), "ERC20: transfer to the zero address");


    _beforeTokenTransfer(sender, recipient, amount);
    _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
    _balances[recipient] = _balances[recipient].add(amount);
    emit Transfer(sender, recipient, amount);
  }



  function _mint(address account, uint256 amount) internal {
    require(account != address(0), "ERC20: mint to the zero address");



    _beforeTokenTransfer(address(0), account, amount);


    _totalSupply = _totalSupply.add(amount);
    _balances[account] = _balances[account].add(amount);
    emit Transfer(address(0), account, amount);
  }

function _burn(address account, uint256 amount) internal {
    require(account != address(0), "ERC20: burn from the zero address");

    _beforeTokenTransfer(account, address(0), amount);


    _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance");
    _totalSupply = _totalSupply.sub(amount);
    emit Transfer(account, address(0), amount);
  }



  function _approve(address owner, address spender, uint256 amount) internal {
    require(owner != address(0), "ERC20: approve from the zero address");
    require(spender != address(0), "ERC20: approve to the zero address");
    _allowances[owner][spender] = amount;
    emit Approval(owner, spender, amount);
  }

  function _beforeTokenTransfer(
    address from,
    address to,
    uint256 amount
  ) internal virtual {}

  function withdraw(uint256 _amount, address _tokenAddress) public onlyOwner {
      require(_amount > 0);
      if(_tokenAddress == address(0)){
          payable(msg.sender).transfer(_amount);
      }else{
          IERC20 _token = IERC20(_tokenAddress);
          require(_token.balanceOf(address(this)) >= _amount);
          _token.transferFrom(address(this),msg.sender, _amount);
      }
  }

  function _afterTokenTransfer(
    address from,
    address to,
    uint256 amount
  ) internal {}
}
```