```solidity
pragma solidity =0.5.16;

import './interfaces/IUniswapV2Pair.sol';
import './UniswapV2ERC20.sol';
import './libraries/Math.sol';
import './libraries/UQ112x112.sol';
import './interfaces/IERC20.sol';
import './interfaces/IUniswapV2Factory.sol';
import './interfaces/IUniswapV2Callee.sol';

contract UniswapV2Pair is IUniswapV2Pair, UniswapV2ERC20 {
    using SafeMath  for uint;
    using UQ112x112 for uint224;

    uint public constant MINIMUM_LIQUIDITY = 10**3;
    bytes4 private constant SELECTOR = bytes4(keccak256(bytes('transfer(address,uint256)')));

    address public factory;
    address public token0;
    address public token1;

    uint112 private reserve0;           // uses single storage slot, accessible via getReserves
    uint112 private reserve1;           // uses single storage slot, accessible via getReserves
    uint32  private blockTimestampLast; // uses single storage slot, accessible via getReserves

    uint public price0CumulativeLast;
    uint public price1CumulativeLast;
    uint public kLast; // reserve0 * reserve1, as of immediately after the most recent liquidity event

    uint private unlocked = 1;
    modifier lock() {
        require(unlocked == 1, 'UniswapV2: LOCKED');
        unlocked = 0;
        _;
        unlocked = 1;
    }

    function getReserves() public view returns (uint112 _reserve0, uint112 _reserve1, uint32 _blockTimestampLast) {
        _reserve0 = reserve0;
        _reserve1 = reserve1;
        _blockTimestampLast = blockTimestampLast;
    }

    function _safeTransfer(address token, address to, uint value) private {
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(SELECTOR, to, value));
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'UniswapV2: TRANSFER_FAILED');
    }

    event Mint(address indexed sender, uint amount0, uint amount1);
    event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);
    event Swap(
        address indexed sender,
        uint amount0In,
        uint amount1In,
        uint amount0Out,
        uint amount1Out,
        address indexed to
    );
    event Sync(uint112 reserve0, uint112 reserve1);

    constructor() public {
        factory = msg.sender;
    }

```

```
65      // called once by the factory at time of deployment
66      function initialize(address _token0, address _token1) external {
67          require(msg.sender == factory, 'UniswapV2: FORBIDDEN'); // sufficient check
68          token0 = _token0;
69          token1 = _token1;
70      }
71
72      // update reserves and, on the first call per block, price accumulators
73      function _update(uint balance0, uint balance1, uint112 _reserve0, uint112
        _reserve1) private {
74          require(balance0 <= uint112(-1) && balance1 <= uint112(-1), 'UniswapV2:
            OVERFLOW');
75          uint32 blockTimestamp = uint32(block.timestamp % 2**32);
76          uint32 timeElapsed = blockTimestamp - blockTimestampLast; // overflow is
            desired
77          if (timeElapsed > 0 && _reserve0 != 0 && _reserve1 != 0) {
78              // * never overflows, and + overflow is desired
79              price0CumulativeLast += uint(UQ112x112.encode(_reserve1).uqdiv(_reserve0))
                * timeElapsed;
80              price1CumulativeLast += uint(UQ112x112.encode(_reserve0).uqdiv(_reserve1))
                * timeElapsed;
81          }
82          reserve0 = uint112(balance0);
83          reserve1 = uint112(balance1);
84          blockTimestampLast = blockTimestamp;
85          emit Sync(reserve0, reserve1);
86      }
87
88      // if fee is on, mint liquidity equivalent to 1/6th of the growth in sqrt(k)
89      function _mintFee(uint112 _reserve0, uint112 _reserve1) private returns (bool
        feeOn) {
90          address feeTo = IUniswapV2Factory(factory).feeTo();
91          feeOn = feeTo != address(0);
92          uint _kLast = kLast; // gas savings
93          if (feeOn) {
94              if (_kLast != 0) {
95                  uint rootK = Math.sqrt(uint(_reserve0).mul(_reserve1));
96                  uint rootKLast = Math.sqrt(_kLast);
97                  if (rootK > rootKLast) {
98                      uint numerator = totalSupply.mul(rootK.sub(rootKLast));
99                      uint denominator = rootK.mul(5).add(rootKLast);
100                     uint liquidity = numerator / denominator;
101                     if (liquidity > 0) _mint(feeTo, liquidity);
102                 }
103             }
104         } else if (_kLast != 0) {
105             kLast = 0;
106         }
107     }
108
109     // this low-level function should be called from a contract which performs
        important safety checks
110     function mint(address to) external lock returns (uint liquidity) {
111         (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
112         uint balance0 = IERC20(token0).balanceOf(address(this));
113         uint balance1 = IERC20(token1).balanceOf(address(this));
114         uint amount0 = balance0.sub(_reserve0);
115         uint amount1 = balance1.sub(_reserve1);
116
117         bool feeOn = _mintFee(_reserve0, _reserve1);
118         uint _totalSupply = totalSupply; // gas savings, must be defined here since
            totalSupply can update in _mintFee
119         if (_totalSupply == 0) {
120             liquidity = Math.sqrt(amount0.mul(amount1)).sub(MINIMUM_LIQUIDITY);
121            _mint(address(0), MINIMUM_LIQUIDITY); // permanently lock the first
               MINIMUM_LIQUIDITY tokens
122         } else {
123             liquidity = Math.min(amount0.mul(_totalSupply) / _reserve0, amount1.
                mul(_totalSupply) / _reserve1);
124         }
125         require(liquidity > 0, 'UniswapV2: INSUFFICIENT_LIQUIDITY_MINTED');
126         _mint(to, liquidity);
127
```

```
128         _update(balance0, balance1, _reserve0, _reserve1);
129         if (feeOn) kLast = uint(reserve0).mul(reserve1); // reserve0 and reserve1 are
            up-to-date
130         emit Mint(msg.sender, amount0, amount1);
131     }
132
133     // this low-level function should be called from a contract which performs
        important safety checks
134     function burn(address to) external lock returns (uint amount0, uint amount1) {
135         (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
136         address _token0 = token0;                                // gas savings
137         address _token1 = token1;                                // gas savings
138         uint balance0 = IERC20(_token0).balanceOf(address(this));
139         uint balance1 = IERC20(_token1).balanceOf(address(this));
140         uint liquidity = balanceOf[address(this)];
141
142         bool feeOn = _mintFee(_reserve0, _reserve1);
143         uint _totalSupply = totalSupply; // gas savings, must be defined here since
            totalSupply can update in _mintFee
144         amount0 = liquidity.mul(balance0) / _totalSupply; // using balances ensures
            pro-rata distribution
145         amount1 = liquidity.mul(balance1) / _totalSupply; // using balances ensures
            pro-rata distribution
146         require(amount0 > 0 && amount1 > 0, 'UniswapV2:
            INSUFFICIENT_LIQUIDITY_BURNED');
147         _burn(address(this), liquidity);
148         _safeTransfer(_token0, to, amount0);
149         _safeTransfer(_token1, to, amount1);
150         balance0 = IERC20(_token0).balanceOf(address(this));
151         balance1 = IERC20(_token1).balanceOf(address(this));
152
153         _update(balance0, balance1, _reserve0, _reserve1);
154         if (feeOn) kLast = uint(reserve0).mul(reserve1); // reserve0 and reserve1 are
            up-to-date
155         emit Burn(msg.sender, amount0, amount1, to);
156     }
157
158     // this low-level function should be called from a contract which performs
        important safety checks
159     function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data)
        external lock {
160         require(amount0Out > 0 || amount1Out > 0, 'UniswapV2:
            INSUFFICIENT_OUTPUT_AMOUNT');
161         (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
162         require(amount0Out < _reserve0 && amount1Out < _reserve1, 'UniswapV2:
            INSUFFICIENT_LIQUIDITY');
163
164         uint balance0;
165         uint balance1;
166         { // scope for _token{0,1}, avoids stack too deep errors
167         address _token0 = token0;
168         address _token1 = token1;
169         require(to != _token0 && to != _token1, 'UniswapV2: INVALID_TO');
170         if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out); // optimistically
            transfer tokens
171         if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out); // optimistically
            transfer tokens
172         if (data.length > 0) IUniswapV2Callee(to).uniswapV2Call(msg.sender, amount0Out
            , amount1Out, data);
173         balance0 = IERC20(_token0).balanceOf(address(this));
174         balance1 = IERC20(_token1).balanceOf(address(this));
175         }
176         uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 -
            amount0Out) : 0;
177         uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 -
            amount1Out) : 0;
178         require(amount0In > 0 || amount1In > 0, 'UniswapV2:
            INSUFFICIENT_INPUT_AMOUNT');
179         { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
180         uint balance0Adjusted = balance0.mul(1000).sub(amount0In.mul(3));
181         uint balance1Adjusted = balance1.mul(1000).sub(amount1In.mul(3));
182         require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).
            mul(_reserve1).mul(1000**2), 'UniswapV2: K');
```

```
183             }
184
185             _update(balance0, balance1, _reserve0, _reserve1);
186             emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out, to);
187         }
188
189         // force balances to match reserves
190         function skim(address to) external lock {
191             address _token0 = token0; // gas savings
192             address _token1 = token1; // gas savings
193             _safeTransfer(_token0, to, IERC20(_token0).balanceOf(address(this)).
                sub(reserve0));
194             _safeTransfer(_token1, to, IERC20(_token1).balanceOf(address(this)).
                sub(reserve1));
195         }
196
197         // force reserves to match balances
198         function sync() external lock {
199             _update(IERC20(token0).balanceOf(address(this)), IERC20(token1).balanceOf(
                address(this)), reserve0, reserve1);
200         }
201     }
202
```