

```
1 /**
2  *Submitted for verification at BscScan.com on
3  2021-05-03
4  */
5 /**
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
```

```
1 /**
2  *Submitted for verification at BscScan.com on
3  2021-05-02
4  */
5 /**
6  KimJongMoon - The first Nuclear Crypto
7
8  KimJongMoon not only has limitless meme token
9  possibilities (probably the most marketable
10 token you could ask for), but it also awards
11 the ARMY of HOLDERS at 1000, 2000, 3000, 4000
12 and 5000 holders
13 milestones.
14
15 Kimy's GENERALS - 1000 holders - 100 Trillion
16 Tokens burnt (10% of total supply)
17 Kimy's COLONELS - 2000 holders - 100 Trillion
18 Tokens burnt (10% of total supply)
19 Kimy's SERGEANTS - 3000 holders - 100 Trillion
20 Tokens burnt (10% of total supply)
21 Kimy's CAPORALS - 4000 holders - 100 Trillion
22 Tokens burnt (10% of total supply)
23 Kimy's SOLDIERS - 5000 holders - 100 Trillion
24 Tokens burnt (10% of total supply)
25
26 So after we establishing our army of 5000 holders,
27 50% of total supply will be burnt!!
28
29 This project was launched as a fairlaunch and
30 will be community run.
31
32
```

```

33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50 Features of DonkeyDong:
51 10% FEES PER TRANSACTION
52 - 5% fee per transaction auto add to the li
liquidity pool
53 - 5% fee per transaction auto distributed t
o holders
54
55 DonkeyDong will start with 0.3 BNB in liqui
dity.
56 25% of total supply will be burnt at launc
h!
57 25% of total supply will be burnt when we r
each 2500 holders!
58
59 Once intial setup is complete, and 2500 hol
ders milestone has been reached,
60 creators will renounce ownership and this t
oken will belong to the community.
61
62 TELEGRAM: https://t.me/DonkeyDongToken
63 WEBSITE: https://donkeydong.info/
64
65 */
66
67 pragma solidity ^0.6.12;
68 // SPDX-License-Identifier: Unlicensed
69 interface IERC20 {
70
71     function totalSupply() external view return

```

```

21
22 Some features of KimJongMoon:
23
24 10% FEES PER TRANSACTION
25 - 5% fee per transaction auto add to the li
liquidity pool
26 - 5% fee per transaction auto distributed t
o holders
27
28
29 KimJongMoon will start with 10 BNB in liqui
dity so no whales will be present.
30 As more holders buy, you can then increase
your buy orders.
31
32 1,000,000,000,000,000 total supply (1 Quadr
illion).
33
34 TELEGRAM: https://t.me/KimJongMoonCoin
35 WEBSITE: https://www.kimjongmoon.net/
36 Reddit : https://www.reddit.com/r/KimJongMo
onToken/
37 Discord : https://discord.com/invite/9WC6Ea
SusA
38
39
40 */
41
42 pragma solidity ^0.6.12;
43 // SPDX-License-Identifier: Unlicensed
44 interface IERC20 {
45
46     function totalSupply() external view return

```

```

s (uint256);
72
73 /**
74  * @dev Returns the amount of tokens owned
  by `account`.
75  */
76  function balanceOf(address account) external
  view returns (uint256);
77
78  /**
79  * @dev Moves `amount` tokens from the call
  er's account to `recipient`.
80  *
81  * Returns a boolean value indicating wheth
  er the operation succeeded.
82  *
83  * Emits a {Transfer} event.
84  */
85  function transfer(address recipient, uint25
  6 amount) external returns (bool);
86
87  /**
88  * @dev Returns the remaining number of tok
  ens that `spender` will be
89  * allowed to spend on behalf of `owner` th
  rough {transferFrom}. This is
90  * zero by default.
91  *
92  * This value changes when {approve} or {tr
  ansferFrom} are called.
93  */
94  function allowance(address owner, address s
  pender) external view returns (uint256);
95
96  /**
97  * @dev Sets `amount` as the allowance of `
  spender` over the caller's tokens.
98  *
99  * Returns a boolean value indicating wheth
  er the operation succeeded.
100  *
101  * IMPORTANT: Beware that changing an allow
  ance with this method brings the risk
102  * that someone may use both the old and th
  e new allowance by unfortunate
103  * transaction ordering. One possible solut
  ion to mitigate this race
104  * condition is to first reduce the spende
  r's allowance to 0 and set the
105  * desired value afterwards:
106  * https://github.com/ethereum/EIPs/issues/
  20#issuecomment-263524729
107  *
108  * Emits an {Approval} event.
109  */
110  function approve(address spender, uint256 a
  mount) external returns (bool);
111
112  /**
113  * @dev Moves `amount` tokens from `sender`
  to `recipient` using the
114  * allowance mechanism. `amount` is then de
  ducted from the caller's
115  * allowance.
116  *
117  * Returns a boolean value indicating wheth

```

```

s (uint256);
47
48 /**
49  * @dev Returns the amount of tokens owned
  by `account`.
50  */
51  function balanceOf(address account) externa
  l view returns (uint256);
52
53  /**
54  * @dev Moves `amount` tokens from the call
  er's account to `recipient`.
55  *
56  * Returns a boolean value indicating wheth
  er the operation succeeded.
57  *
58  * Emits a {Transfer} event.
59  */
60  function transfer(address recipient, uint25
  6 amount) external returns (bool);
61
62  /**
63  * @dev Returns the remaining number of tok
  ens that `spender` will be
64  * allowed to spend on behalf of `owner` th
  rough {transferFrom}. This is
65  * zero by default.
66  *
67  * This value changes when {approve} or {tr
  ansferFrom} are called.
68  */
69  function allowance(address owner, address s
  pender) external view returns (uint256);
70
71  /**
72  * @dev Sets `amount` as the allowance of `
  spender` over the caller's tokens.
73  *
74  * Returns a boolean value indicating wheth
  er the operation succeeded.
75  *
76  * IMPORTANT: Beware that changing an allow
  ance with this method brings the risk
77  * that someone may use both the old and th
  e new allowance by unfortunate
78  * transaction ordering. One possible solut
  ion to mitigate this race
79  * condition is to first reduce the spende
  r's allowance to 0 and set the
80  * desired value afterwards:
81  * https://github.com/ethereum/EIPs/issues/
  20#issuecomment-263524729
82  *
83  * Emits an {Approval} event.
84  */
85  function approve(address spender, uint256 a
  mount) external returns (bool);
86
87  /**
88  * @dev Moves `amount` tokens from `sender`
  to `recipient` using the
89  * allowance mechanism. `amount` is then de
  ducted from the caller's
90  * allowance.
91  *
92  * Returns a boolean value indicating wheth

```

```

er the operation succeeded.
118     *
119     * Emits a {Transfer} event.
120     */
121     function transferFrom(address sender, address
recipient, uint256 amount) external returns
        (bool);
122
123     /**
124     * @dev Emitted when `value` tokens are mov
ed from one account (`from`) to
125     * another (`to`).
126     *
127     * Note that `value` may be zero.
128     */
129     event Transfer(address indexed from, address
indexed to, uint256 value);
130
131     /**
132     * @dev Emitted when the allowance of a `sp
ender` for an `owner` is set by
133     * a call to {approve}. `value` is the new
allowance.
134     */
135     event Approval(address indexed owner, address
indexed spender, uint256 value);
136 }
137
138
139
140 /**
141 * @dev Wrappers over Solidity's arithmetic ope
rations with added overflow
142 * checks.
143 *
144 * Arithmetic operations in Solidity wrap on ov
erflow. This can easily result
145 * in bugs, because programmers usually assume
that an overflow raises an
146 * error, which is the standard behavior in hig
h level programming languages.
147 * `SafeMath` restores this intuition by revert
ing the transaction when an
148 * operation overflows.
149 *
150 * Using this library instead of the unchecked
operations eliminates an entire
151 * class of bugs, so it's recommended to use it
always.
152 */
153
154 library SafeMath {
155     /**
156     * @dev Returns the addition of two unsigne
d integers, reverting on
157     * overflow.
158     *
159     * Counterpart to Solidity's `+` operator.
160     *
161     * Requirements:
162     *
163     * - Addition cannot overflow.
164     */
165     function add(uint256 a, uint256 b) internal
pure returns (uint256) {
166         uint256 c = a + b;
167

```

```

er the operation succeeded.
93     *
94     * Emits a {Transfer} event.
95     */
96     function transferFrom(address sender, address
ss recipient, uint256 amount) external returns
        (bool);
97
98     /**
99     * @dev Emitted when `value` tokens are mov
ed from one account (`from`) to
100     * another (`to`).
101     *
102     * Note that `value` may be zero.
103     */
104     event Transfer(address indexed from, address
s indexed to, uint256 value);
105
106     /**
107     * @dev Emitted when the allowance of a `sp
ender` for an `owner` is set by
108     * a call to {approve}. `value` is the new
allowance.
109     */
110     event Approval(address indexed owner, address
ss indexed spender, uint256 value);
111 }
112
113
114
115 /**
116 * @dev Wrappers over Solidity's arithmetic ope
rations with added overflow
117 * checks.
118 *
119 * Arithmetic operations in Solidity wrap on ov
erflow. This can easily result
120 * in bugs, because programmers usually assume
that an overflow raises an
121 * error, which is the standard behavior in hig
h level programming languages.
122 * `SafeMath` restores this intuition by revert
ing the transaction when an
123 * operation overflows.
124 *
125 * Using this library instead of the unchecked
operations eliminates an entire
126 * class of bugs, so it's recommended to use it
always.
127 */
128
129 library SafeMath {
130     /**
131     * @dev Returns the addition of two unsigne
d integers, reverting on
132     * overflow.
133     *
134     * Counterpart to Solidity's `+` operator.
135     *
136     * Requirements:
137     *
138     * - Addition cannot overflow.
139     */
140     function add(uint256 a, uint256 b) internal
pure returns (uint256) {
141         uint256 c = a + b;
142

```

```

        require(c >= a, "SafeMath: addition ove
rflow");
168
169     return c;
170 }
171
172 /**
173  * @dev Returns the subtraction of two unsi
gned integers, reverting on
174  * overflow (when the result is negative).
175  *
176  * Counterpart to Solidity's '-' operator.
177  *
178  * Requirements:
179  *
180  * - Subtraction cannot overflow.
181  */
182     function sub(uint256 a, uint256 b) internal
pure returns (uint256) {
183         return sub(a, b, "SafeMath: subtraction
overflow");
184     }
185
186 /**
187  * @dev Returns the subtraction of two unsi
gned integers, reverting with custom message on
188  * overflow (when the result is negative).
189  *
190  * Counterpart to Solidity's '-' operator.
191  *
192  * Requirements:
193  *
194  * - Subtraction cannot overflow.
195  */
196     function sub(uint256 a, uint256 b, string m
emory errorMessage) internal pure returns (uint
256) {
197         require(b <= a, errorMessage);
198         uint256 c = a - b;
199
200         return c;
201     }
202
203 /**
204  * @dev Returns the multiplication of two u
nsigned integers, reverting on
205  * overflow.
206  *
207  * Counterpart to Solidity's '*' operator.
208  *
209  * Requirements:
210  *
211  * - Multiplication cannot overflow.
212  */
213     function mul(uint256 a, uint256 b) internal
pure returns (uint256) {
214         // Gas optimization: this is cheaper th
an requiring 'a' not being zero, but the
215         // benefit is lost if 'b' is also teste
d.
216         // See: https://github.com/OpenZeppeli
n/openzeppelin-contracts/pull/522
217         if (a == 0) {
218             return 0;
219         }
220
221         uint256 c = a * b;

```

```

        require(c >= a, "SafeMath: addition ove
rflow");
143
144     return c;
145 }
146
147 /**
148  * @dev Returns the subtraction of two unsi
gned integers, reverting on
149  * overflow (when the result is negative).
150  *
151  * Counterpart to Solidity's '-' operator.
152  *
153  * Requirements:
154  *
155  * - Subtraction cannot overflow.
156  */
157     function sub(uint256 a, uint256 b) internal
pure returns (uint256) {
158         return sub(a, b, "SafeMath: subtraction
overflow");
159     }
160
161 /**
162  * @dev Returns the subtraction of two unsi
gned integers, reverting with custom message on
163  * overflow (when the result is negative).
164  *
165  * Counterpart to Solidity's '-' operator.
166  *
167  * Requirements:
168  *
169  * - Subtraction cannot overflow.
170  */
171     function sub(uint256 a, uint256 b, string m
emory errorMessage) internal pure returns (uint
256) {
172         require(b <= a, errorMessage);
173         uint256 c = a - b;
174
175         return c;
176     }
177
178 /**
179  * @dev Returns the multiplication of two u
nsigned integers, reverting on
180  * overflow.
181  *
182  * Counterpart to Solidity's '*' operator.
183  *
184  * Requirements:
185  *
186  * - Multiplication cannot overflow.
187  */
188     function mul(uint256 a, uint256 b) internal
pure returns (uint256) {
189         // Gas optimization: this is cheaper th
an requiring 'a' not being zero, but the
190         // benefit is lost if 'b' is also teste
d.
191         // See: https://github.com/OpenZeppeli
n/openzeppelin-contracts/pull/522
192         if (a == 0) {
193             return 0;
194         }
195
196         uint256 c = a * b;

```

```

222         require(c / a == b, "SafeMath: multipli
cation overflow");
223
224         return c;
225     }
226
227     /**
228      * @dev Returns the integer division of two
unsigned integers. Reverts on
229      * division by zero. The result is rounded
towards zero.
230      *
231      * Counterpart to Solidity's `/` operator.
Note: this function uses a
232      * `revert` opcode (which leaves remaining
gas untouched) while Solidity
233      * uses an invalid opcode to revert (consum
ing all remaining gas).
234      *
235      * Requirements:
236      *
237      * - The divisor cannot be zero.
238      */
239     function div(uint256 a, uint256 b) internal
pure returns (uint256) {
240         return div(a, b, "SafeMath: division by
zero");
241     }
242
243     /**
244      * @dev Returns the integer division of two
unsigned integers. Reverts with custom message
on
245      * division by zero. The result is rounded
towards zero.
246      *
247      * Counterpart to Solidity's `/` operator.
Note: this function uses a
248      * `revert` opcode (which leaves remaining
gas untouched) while Solidity
249      * uses an invalid opcode to revert (consum
ing all remaining gas).
250      *
251      * Requirements:
252      *
253      * - The divisor cannot be zero.
254      */
255     function div(uint256 a, uint256 b, string m
emory errorMessage) internal pure returns (uint
256) {
256         require(b > 0, errorMessage);
257         uint256 c = a / b;
258         // assert(a == b * c + a % b); // There
is no case in which this doesn't hold
259
260         return c;
261     }
262
263     /**
264      * @dev Returns the remainder of dividing t
wo unsigned integers. (unsigned integer modul
o),
265      * Reverts when dividing by zero.
266      *
267      * Counterpart to Solidity's `%` operator.
This function uses a `revert`
268

```

```

197         require(c / a == b, "SafeMath: multipli
cation overflow");
198
199         return c;
200     }
201
202     /**
203      * @dev Returns the integer division of two
unsigned integers. Reverts on
204      * division by zero. The result is rounded
towards zero.
205      *
206      * Counterpart to Solidity's `/` operator.
Note: this function uses a
207      * `revert` opcode (which leaves remaining
gas untouched) while Solidity
208      * uses an invalid opcode to revert (consum
ing all remaining gas).
209      *
210      * Requirements:
211      *
212      * - The divisor cannot be zero.
213      */
214     function div(uint256 a, uint256 b) internal
pure returns (uint256) {
215         return div(a, b, "SafeMath: division by
zero");
216     }
217
218     /**
219      * @dev Returns the integer division of two
unsigned integers. Reverts with custom message
on
220      * division by zero. The result is rounded
towards zero.
221      *
222      * Counterpart to Solidity's `/` operator.
Note: this function uses a
223      * `revert` opcode (which leaves remaining
gas untouched) while Solidity
224      * uses an invalid opcode to revert (consum
ing all remaining gas).
225      *
226      * Requirements:
227      *
228      * - The divisor cannot be zero.
229      */
230     function div(uint256 a, uint256 b, string m
emory errorMessage) internal pure returns (uint
231) {
231         require(b > 0, errorMessage);
232         uint256 c = a / b;
233         // assert(a == b * c + a % b); // There
is no case in which this doesn't hold
234
235         return c;
236     }
237
238     /**
239      * @dev Returns the remainder of dividing t
wo unsigned integers. (unsigned integer modul
o),
240      * Reverts when dividing by zero.
241      *
242      * Counterpart to Solidity's `%` operator.
This function uses a `revert`
243

```

```

        * opcode (which leaves remaining gas untou
ched) while Solidity uses an
269     * invalid opcode to revert (consuming all
remaining gas).
270     *
271     * Requirements:
272     *
273     * - The divisor cannot be zero.
274     */
275     function mod(uint256 a, uint256 b) internal
pure returns (uint256) {
276         return mod(a, b, "SafeMath: modulo by z
ero");
277     }
278
279     /**
280     * @dev Returns the remainder of dividing t
wo unsigned integers. (unsigned integer modul
o),
281     * Reverts with custom message when dividin
g by zero.
282     *
283     * Counterpart to Solidity's `%` operator.
This function uses a `revert`
284     * opcode (which leaves remaining gas untou
ched) while Solidity uses an
285     * invalid opcode to revert (consuming all
remaining gas).
286     *
287     * Requirements:
288     *
289     * - The divisor cannot be zero.
290     */
291     function mod(uint256 a, uint256 b, string m
emory errorMessage) internal pure returns (uint
256) {
292         require(b != 0, errorMessage);
293         return a % b;
294     }
295 }
296
297 abstract contract Context {
298     function _msgSender() internal view virtual
returns (address payable) {
299         return msg.sender;
300     }
301
302     function _msgData() internal view virtual r
eturns (bytes memory) {
303         this; // silence state mutability warni
ng without generating bytecode - see https://gi
thub.com/ethereum/solidity/issues/2691
304         return msg.data;
305     }
306 }
307
308
309 /**
310 * @dev Collection of functions related to the
address type
311 */
312 library Address {
313     /**
314     * @dev Returns true if `account` is a cont
ract.
315     *
316

```

```

        * opcode (which leaves remaining gas untou
ched) while Solidity uses an
244     * invalid opcode to revert (consuming all
remaining gas).
245     *
246     * Requirements:
247     *
248     * - The divisor cannot be zero.
249     */
250     function mod(uint256 a, uint256 b) internal
pure returns (uint256) {
251         return mod(a, b, "SafeMath: modulo by z
ero");
252     }
253
254     /**
255     * @dev Returns the remainder of dividing t
wo unsigned integers. (unsigned integer modul
o),
256     * Reverts with custom message when dividin
g by zero.
257     *
258     * Counterpart to Solidity's `%` operator.
This function uses a `revert`
259     * opcode (which leaves remaining gas untou
ched) while Solidity uses an
260     * invalid opcode to revert (consuming all
remaining gas).
261     *
262     * Requirements:
263     *
264     * - The divisor cannot be zero.
265     */
266     function mod(uint256 a, uint256 b, string m
emory errorMessage) internal pure returns (uint
256) {
267         require(b != 0, errorMessage);
268         return a % b;
269     }
270 }
271
272 abstract contract Context {
273     function _msgSender() internal view virtual
returns (address payable) {
274         return msg.sender;
275     }
276
277     function _msgData() internal view virtual r
eturns (bytes memory) {
278         this; // silence state mutability warni
ng without generating bytecode - see https://gi
thub.com/ethereum/solidity/issues/2691
279         return msg.data;
280     }
281 }
282
283
284 /**
285 * @dev Collection of functions related to the
address type
286 */
287 library Address {
288     /**
289     * @dev Returns true if `account` is a cont
ract.
290     *
291

```

```

    * [IMPORTANT]
317 * ====
318 * It is unsafe to assume that an address f
or which this function returns
319 * false is an externally-owned account (EO
A) and not a contract.
320 *
321 * Among others, `isContract` will return f
alse for the following
322 * types of addresses:
323 *
324 * - an externally-owned account
325 * - a contract in construction
326 * - an address where a contract will be c
reated
327 * - an address where a contract lived, bu
t was destroyed
328 * ====
329 */
330 function isContract(address account) intern
al view returns (bool) {
331     // According to EIP-1052, 0x0 is the va
lue returned for not-yet created accounts
332     // and 0xc5d2460186f7233c927e7db2dcc703
c0e500b653ca82273b7bfad8045d85a470 is returned
333     // for accounts without code, i.e. `kec
cak256(')`
334     bytes32 codehash;
335     bytes32 accountHash = 0xc5d2460186f7233
c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a4
70;
336     // solhint-disable-next-line no-inline-
assembly
337     assembly { codehash := extcodehash(acco
unt) }
338     return (codehash != accountHash && code
hash != 0x0);
339 }
340
341 /**
342  * @dev Replacement for Solidity's `transfe
r`: sends `amount` wei to
343  * `recipient`, forwarding all available ga
s and reverting on errors.
344  *
345  * https://eips.ethereum.org/EIPS/eip-1884
[EIP1884] increases the gas cost
346  * of certain opcodes, possibly making cont
racts go over the 2300 gas limit
347  * imposed by `transfer`, making them unabl
e to receive funds via
348  * `transfer`. {sendValue} removes this lim
itation.
349  *
350  * https://diligence.consensys.net/posts/20
19/09/stop-using-soliditys-transfer-now/[Learn
more].
351  *
352  * IMPORTANT: because control is transferre
d to `recipient`, care must be
353  * taken to not create reentrancy vulnerabi
lities. Consider using
354  * {ReentrancyGuard} or the
355  * https://solidity.readthedocs.io/en/v0.5.
11/security-considerations.html#use-the-checks-

```

```

    * [IMPORTANT]
292 * ====
293 * It is unsafe to assume that an address f
or which this function returns
294 * false is an externally-owned account (EO
A) and not a contract.
295 *
296 * Among others, `isContract` will return f
alse for the following
297 * types of addresses:
298 *
299 * - an externally-owned account
300 * - a contract in construction
301 * - an address where a contract will be c
reated
302 * - an address where a contract lived, bu
t was destroyed
303 * ====
304 */
305 function isContract(address account) intern
al view returns (bool) {
306     // According to EIP-1052, 0x0 is the va
lue returned for not-yet created accounts
307     // and 0xc5d2460186f7233c927e7db2dcc703
c0e500b653ca82273b7bfad8045d85a470 is returned
308     // for accounts without code, i.e. `kec
cak256(')`
309     bytes32 codehash;
310     bytes32 accountHash = 0xc5d2460186f7233
c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a4
70;
311     // solhint-disable-next-line no-inline-
assembly
312     assembly { codehash := extcodehash(acco
unt) }
313     return (codehash != accountHash && code
hash != 0x0);
314 }
315
316 /**
317  * @dev Replacement for Solidity's `transfe
r`: sends `amount` wei to
318  * `recipient`, forwarding all available ga
s and reverting on errors.
319  *
320  * https://eips.ethereum.org/EIPS/eip-1884
[EIP1884] increases the gas cost
321  * of certain opcodes, possibly making cont
racts go over the 2300 gas limit
322  * imposed by `transfer`, making them unabl
e to receive funds via
323  * `transfer`. {sendValue} removes this lim
itation.
324  *
325  * https://diligence.consensys.net/posts/20
19/09/stop-using-soliditys-transfer-now/[Learn
more].
326  *
327  * IMPORTANT: because control is transferre
d to `recipient`, care must be
328  * taken to not create reentrancy vulnerabi
lities. Consider using
329  * {ReentrancyGuard} or the
330  * https://solidity.readthedocs.io/en/v0.5.
11/security-considerations.html#use-the-checks-

```



```

effects-interactions-pattern[checks-effects-interactions pattern].
356 */
357 function sendValue(address payable recipient, uint256 amount) internal {
358     require(address(this).balance >= amount, "Address: insufficient balance");
359
360     // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
361     (bool success, ) = recipient.call{ value: amount }("");
362     require(success, "Address: unable to send value, recipient may have reverted");
363 }
364
365 /**
366  * @dev Performs a Solidity function call using a low level `call`. A
367  * plain `call` is an unsafe replacement for a function call: use this
368  * function instead.
369  *
370  * If `target` reverts with a revert reason, it is bubbled up by this
371  * function (like regular Solidity function calls).
372  *
373  * Returns the raw returned data. To convert to the expected return value,
374  * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding-and-decoding-functions
375  * [abi.decode].
376  *
377  * Requirements:
378  *
379  * - `target` must be a contract.
380  * - calling `target` with `data` must not revert.
381  *
382  * _Available since v3.1._
383  */
384 function functionCall(address target, bytes memory data) internal returns (bytes memory) {
385     return functionCall(target, data, "Address: low-level call failed");
386 }
387
388 /**
389  * @dev Same as {xref-Address-functionCall-address-bytes-}[functionCall], but with
390  * `errorMessage` as a fallback revert reason when `target` reverts.
391  *
392  * _Available since v3.1._
393  */
394 function functionCall(address target, bytes memory data, string memory errorMessage) internal returns (bytes memory) {
395     return _functionCallWithValue(target, data, 0, errorMessage);
396 }
397
398 /**
399  * @dev Same as {xref-Address-functionCall-

```

```

effects-interactions-pattern[checks-effects-interactions pattern].
331 */
332 function sendValue(address payable recipient, uint256 amount) internal {
333     require(address(this).balance >= amount, "Address: insufficient balance");
334
335     // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
336     (bool success, ) = recipient.call{ value: amount }("");
337     require(success, "Address: unable to send value, recipient may have reverted");
338 }
339
340 /**
341  * @dev Performs a Solidity function call using a low level `call`. A
342  * plain `call` is an unsafe replacement for a function call: use this
343  * function instead.
344  *
345  * If `target` reverts with a revert reason, it is bubbled up by this
346  * function (like regular Solidity function calls).
347  *
348  * Returns the raw returned data. To convert to the expected return value,
349  * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding-and-decoding-functions
350  * [abi.decode].
351  *
352  * Requirements:
353  *
354  * - `target` must be a contract.
355  * - calling `target` with `data` must not revert.
356  *
357  * _Available since v3.1._
358  */
359 function functionCall(address target, bytes memory data) internal returns (bytes memory) {
360     return functionCall(target, data, "Address: low-level call failed");
361 }
362
363 /**
364  * @dev Same as {xref-Address-functionCall-address-bytes-}[functionCall], but with
365  * `errorMessage` as a fallback revert reason when `target` reverts.
366  *
367  * _Available since v3.1._
368  */
369 function functionCall(address target, bytes memory data, string memory errorMessage) internal returns (bytes memory) {
370     return _functionCallWithValue(target, data, 0, errorMessage);
371 }
372
373 /**
374  * @dev Same as {xref-Address-functionCall-

```

```

address-bytes-}[`functionCall`],
399     * but also transferring `value` wei to `target`.
400     *
401     * Requirements:
402     *
403     * - the calling contract must have an ETH
      balance of at least `value`.
404     * - the called Solidity function must be `payable`.
405     *
406     * _Available since v3.1._
407     */
408     function functionCallWithValue(address target, bytes memory data, uint256 value) internal
      returns (bytes memory) {
409         return functionCallWithValue(target, data, value, "Address: low-level call with value
      failed");
410     }
411
412     /**
413     * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithValue`], but
414     * with `errorMessage` as a fallback revert reason when `target` reverts.
415     *
416     * _Available since v3.1._
417     */
418     function functionCallWithValue(address target, bytes memory data, uint256 value, string memory errorMessage) internal returns (bytes memory) {
419         require(address(this).balance >= value, "Address: insufficient balance for call");
420         return _functionCallWithValue(target, data, value, errorMessage);
421     }
422
423     function _functionCallWithValue(address target, bytes memory data, uint256 weiValue, string memory errorMessage) private returns (bytes memory) {
424         require(isContract(target), "Address: call to non-contract");
425
426         // solhint-disable-next-line avoid-low-level-calls
427         (bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
428         if (success) {
429             return returndata;
430         } else {
431             // Look for revert reason and bubble it up if present
432             if (returndata.length > 0) {
433                 // The easiest way to bubble the revert reason is using memory via assembly
434
435                 // solhint-disable-next-line no-inline-assembly
436                 assembly {
437                     let returndata_size := mload(returndata)
438                     revert(add(32, returndata),

```

```

address-bytes-}[`functionCall`],
374     * but also transferring `value` wei to `target`.
375     *
376     * Requirements:
377     *
378     * - the calling contract must have an ETH
      balance of at least `value`.
379     * - the called Solidity function must be `payable`.
380     *
381     * _Available since v3.1._
382     */
383     function functionCallWithValue(address target, bytes memory data, uint256 value) internal
      returns (bytes memory) {
384         return functionCallWithValue(target, data, value, "Address: low-level call with value
      failed");
385     }
386
387     /**
388     * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithValue`], but
389     * with `errorMessage` as a fallback revert reason when `target` reverts.
390     *
391     * _Available since v3.1._
392     */
393     function functionCallWithValue(address target, bytes memory data, uint256 value, string memory errorMessage) internal returns (bytes memory) {
394         require(address(this).balance >= value, "Address: insufficient balance for call");
395         return _functionCallWithValue(target, data, value, errorMessage);
396     }
397
398     function _functionCallWithValue(address target, bytes memory data, uint256 weiValue, string memory errorMessage) private returns (bytes memory) {
399         require(isContract(target), "Address: call to non-contract");
400
401         // solhint-disable-next-line avoid-low-level-calls
402         (bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
403         if (success) {
404             return returndata;
405         } else {
406             // Look for revert reason and bubble it up if present
407             if (returndata.length > 0) {
408                 // The easiest way to bubble the revert reason is using memory via assembly
409
410                 // solhint-disable-next-line no-inline-assembly
411                 assembly {
412                     let returndata_size := mload(returndata)
413                     revert(add(32, returndata),

```

```

        returndata_size)
439     }
440     } else {
441         revert(errorMessage);
442     }
443 }
444 }
445 }
446
447 /**
448  * @dev Contract module which provides a basic
449  * access control mechanism, where
450  * there is an account (an owner) that can be g
451  * ranted exclusive access to
452  * specific functions.
453  *
454  * By default, the owner account will be the on
455  * e that deploys the contract. This
456  * can later be changed with {transferOwnersh
457  * ip}.
458  *
459  * This module is used through inheritance. It
460  * will make available the modifier
461  * `onlyOwner`, which can be applied to your fu
462  * nctions to restrict their use to
463  * the owner.
464  */
465 contract Ownable is Context {
466     address private _owner;
467     address private _previousOwner;
468     uint256 private _lockTime;
469
470     event OwnershipTransferred(address indexed
471     previousOwner, address indexed newOwner);
472
473     /**
474      * @dev Initializes the contract setting th
475      * e deployer as the initial owner.
476      */
477     constructor () internal {
478         address msgSender = _msgSender();
479         _owner = msgSender;
480         emit OwnershipTransferred(address(0), m
481         sgSender);
482     }
483
484     /**
485      * @dev Returns the address of the current
486      * owner.
487      */
488     function owner() public view returns (addre
489     ss) {
490         return _owner;
491     }
492
493     /**
494      * @dev Throws if called by any account oth
495      * er than the owner.
496      */
497     modifier onlyOwner() {
498         require(_owner == _msgSender(), "Ownabl
499         e: caller is not the owner");
500         _;
501     }
502
503     /**
504      * @dev Leaves the contract without owner.

```

```

        returndata_size)
414     }
415     } else {
416         revert(errorMessage);
417     }
418 }
419 }
420 }
421
422 /**
423  * @dev Contract module which provides a basic
424  * access control mechanism, where
425  * there is an account (an owner) that can be g
426  * ranted exclusive access to
427  * specific functions.
428  *
429  * By default, the owner account will be the on
430  * e that deploys the contract. This
431  * can later be changed with {transferOwnersh
432  * ip}.
433  *
434  * This module is used through inheritance. It
435  * will make available the modifier
436  * `onlyOwner`, which can be applied to your fu
437  * nctions to restrict their use to
438  * the owner.
439  */
440 contract Ownable is Context {
441     address private _owner;
442     address private _previousOwner;
443     uint256 private _lockTime;
444
445     event OwnershipTransferred(address indexed
446     previousOwner, address indexed newOwner);
447
448     /**
449      * @dev Initializes the contract setting th
450      * e deployer as the initial owner.
451      */
452     constructor () internal {
453         address msgSender = _msgSender();
454         _owner = msgSender;
455         emit OwnershipTransferred(address(0), m
456         sgSender);
457     }
458
459     /**
460      * @dev Returns the address of the current
461      * owner.
462      */
463     function owner() public view returns (addre
464     ss) {
465         return _owner;
466     }
467
468     /**
469      * @dev Throws if called by any account oth
470      * er than the owner.
471      */
472     modifier onlyOwner() {
473         require(_owner == _msgSender(), "Ownabl
474         e: caller is not the owner");
475         _;
476     }
477
478     /**
479      * @dev Leaves the contract without owner.

```

```

It will not be possible to call
492 * `onlyOwner` functions anymore. Can only
    be called by the current owner.
493 *
494 * NOTE: Renouncing ownership will leave the contract without an owner,
    * thereby removing any functionality that
    is only available to the owner.
496 */
497 function renounceOwnership() public virtual
    onlyOwner {
498     emit OwnershipTransferred(_owner, address(0));
499     _owner = address(0);
500 }
501
502 /**
503  * @dev Transfers ownership of the contract
    to a new account (`newOwner`).
504  * Can only be called by the current owner.
505  */
506 function transferOwnership(address newOwner) public virtual onlyOwner {
507     require(newOwner != address(0), "Ownable: new owner is the zero address");
508     emit OwnershipTransferred(_owner, newOwner);
509     _owner = newOwner;
510 }
511
512 function getUnlockTime() public view returns (uint256) {
513     return _lockTime;
514 }
515
516 //Locks the contract for owner for the amount of time provided
517 function lock(uint256 time) public virtual onlyOwner {
518     _previousOwner = _owner;
519     _owner = address(0);
520     _lockTime = now + time;
521     emit OwnershipTransferred(_owner, address(0));
522 }
523
524 //Unlocks the contract for owner when _lockTime is exceeded
525 function unlock() public virtual {
526     require(_previousOwner == msg.sender, "You don't have permission to unlock");
527     require(now > _lockTime, "Contract is locked until 7 days");
528     emit OwnershipTransferred(_owner, _previousOwner);
529     _owner = _previousOwner;
530 }
531 }
532
533 // pragma solidity >=0.5.0;
534
535 interface IUniswapV2Factory {
536     event PairCreated(address indexed token0, address indexed token1, address pair, uint);
537
538     function feeTo() external view returns (address);

```

```

It will not be possible to call
467 * `onlyOwner` functions anymore. Can only
    be called by the current owner.
468 *
469 * NOTE: Renouncing ownership will leave the contract without an owner,
    * thereby removing any functionality that
    is only available to the owner.
471 */
472 function renounceOwnership() public virtual
    onlyOwner {
473     emit OwnershipTransferred(_owner, address(0));
474     _owner = address(0);
475 }
476
477 /**
478  * @dev Transfers ownership of the contract
    to a new account (`newOwner`).
479  * Can only be called by the current owner.
480  */
481 function transferOwnership(address newOwner) public virtual onlyOwner {
482     require(newOwner != address(0), "Ownable: new owner is the zero address");
483     emit OwnershipTransferred(_owner, newOwner);
484     _owner = newOwner;
485 }
486
487 function getUnlockTime() public view returns (uint256) {
488     return _lockTime;
489 }
490
491 //Locks the contract for owner for the amount of time provided
492 function lock(uint256 time) public virtual onlyOwner {
493     _previousOwner = _owner;
494     _owner = address(0);
495     _lockTime = now + time;
496     emit OwnershipTransferred(_owner, address(0));
497 }
498
499 //Unlocks the contract for owner when _lockTime is exceeded
500 function unlock() public virtual {
501     require(_previousOwner == msg.sender, "You don't have permission to unlock");
502     require(now > _lockTime, "Contract is locked until 7 days");
503     emit OwnershipTransferred(_owner, _previousOwner);
504     _owner = _previousOwner;
505 }
506 }
507
508 // pragma solidity >=0.5.0;
509
510 interface IUniswapV2Factory {
511     event PairCreated(address indexed token0, address indexed token1, address pair, uint);
512
513     function feeTo() external view returns (address);

```

```

539     function feeToSetter() external view return
s (address);
540
541     function getPair(address tokenA, address to
kenB) external view returns (address pair);
542     function allPairs(uint) external view retur
ns (address pair);
543     function allPairsLength() external view ret
urns (uint);
544
545     function createPair(address tokenA, address
tokenB) external returns (address pair);
546
547     function setFeeTo(address) external;
548     function setFeeToSetter(address) external;
549 }
550
551
552 // pragma solidity >=0.5.0;
553
554 interface IUniswapV2Pair {
555     event Approval(address indexed owner, addre
ss indexed spender, uint value);
556     event Transfer(address indexed from, addres
s indexed to, uint value);
557
558     function name() external pure returns (stri
ng memory);
559     function symbol() external pure returns (st
ring memory);
560     function decimals() external pure returns
(uint8);
561     function totalSupply() external view return
s (uint);
562     function balanceOf(address owner) external
view returns (uint);
563     function allowance(address owner, address s
pender) external view returns (uint);
564
565     function approve(address spender, uint valu
e) external returns (bool);
566     function transfer(address to, uint value) e
xternal returns (bool);
567     function transferFrom(address from, address
to, uint value) external returns (bool);
568
569     function DOMAIN_SEPARATOR() external view r
eturns (bytes32);
570     function PERMIT_TYPEHASH() external pure re
turns (bytes32);
571     function nonces(address owner) external vie
w returns (uint);
572
573     function permit(address owner, address spen
der, uint value, uint deadline, uint8 v, bytes3
2 r, bytes32 s) external;
574
575     event Mint(address indexed sender, uint amo
unt0, uint amount1);
576     event Burn(address indexed sender, uint amo
unt0, uint amount1, address indexed to);
577     event Swap(
578         address indexed sender,
579         uint amount0In,
580         uint amount1In,
581         uint amount0Out,
582         uint amount1Out,
583         address indexed to

```

```

514     function feeToSetter() external view return
s (address);
515
516     function getPair(address tokenA, address to
kenB) external view returns (address pair);
517     function allPairs(uint) external view retur
ns (address pair);
518     function allPairsLength() external view ret
urns (uint);
519
520     function createPair(address tokenA, address
tokenB) external returns (address pair);
521
522     function setFeeTo(address) external;
523     function setFeeToSetter(address) external;
524 }
525
526
527 // pragma solidity >=0.5.0;
528
529 interface IUniswapV2Pair {
530     event Approval(address indexed owner, addre
ss indexed spender, uint value);
531     event Transfer(address indexed from, addres
s indexed to, uint value);
532
533     function name() external pure returns (stri
ng memory);
534     function symbol() external pure returns (st
ring memory);
535     function decimals() external pure returns
(uint8);
536     function totalSupply() external view return
s (uint);
537     function balanceOf(address owner) external
view returns (uint);
538     function allowance(address owner, address s
pender) external view returns (uint);
539
540     function approve(address spender, uint valu
e) external returns (bool);
541     function transfer(address to, uint value) e
xternal returns (bool);
542     function transferFrom(address from, address
to, uint value) external returns (bool);
543
544     function DOMAIN_SEPARATOR() external view r
eturns (bytes32);
545     function PERMIT_TYPEHASH() external pure re
turns (bytes32);
546     function nonces(address owner) external vie
w returns (uint);
547
548     function permit(address owner, address spen
der, uint value, uint deadline, uint8 v, bytes3
2 r, bytes32 s) external;
549
550     event Mint(address indexed sender, uint amo
unt0, uint amount1);
551     event Burn(address indexed sender, uint amo
unt0, uint amount1, address indexed to);
552     event Swap(
553         address indexed sender,
554         uint amount0In,
555         uint amount1In,
556         uint amount0Out,
557         uint amount1Out,
558         address indexed to

```

```

584     );
585     event Sync(uint112 reserve0, uint112 reserv
e1);
586
587     function MINIMUM_LIQUIDITY() external pure
returns (uint);
588     function factory() external view returns (a
ddress);
589     function token0() external view returns (ad
dress);
590     function token1() external view returns (ad
dress);
591     function getReserves() external view return
s (uint112 reserve0, uint112 reserve1, uint32 b
lockTimestampLast);
592     function price0CumulativeLast() external vi
ew returns (uint);
593     function price1CumulativeLast() external vi
ew returns (uint);
594     function kLast() external view returns (uin
t);
595
596     function mint(address to) external returns
(uint liquidity);
597     function burn(address to) external returns
(uint amount0, uint amount1);
598     function swap(uint amount0Out, uint amount1
Out, address to, bytes calldata data) external;
599     function skim(address to) external;
600     function sync() external;
601
602     function initialize(address, address) exter
nal;
603 }
604
605 // pragma solidity >=0.6.2;
606
607 interface IUniswapV2Router01 {
608     function factory() external pure returns (a
ddress);
609     function WETH() external pure returns (addr
ess);
610
611     function addLiquidity(
612         address tokenA,
613         address tokenB,
614         uint amountADesired,
615         uint amountBDesired,
616         uint amountAMin,
617         uint amountBMin,
618         address to,
619         uint deadline
620     ) external returns (uint amountA, uint amou
ntB, uint liquidity);
621     function addLiquidityETH(
622         address token,
623         uint amountTokenDesired,
624         uint amountTokenMin,
625         uint amountETHMin,
626         address to,
627         uint deadline
628     ) external payable returns (uint amountToke
n, uint amountETH, uint liquidity);
629     function removeLiquidity(
630         address tokenA,
631         address tokenB,
632         uint liquidity,

```

```

559     );
560     event Sync(uint112 reserve0, uint112 reserv
e1);
561
562     function MINIMUM_LIQUIDITY() external pure
returns (uint);
563     function factory() external view returns (a
ddress);
564     function token0() external view returns (ad
dress);
565     function token1() external view returns (ad
dress);
566     function getReserves() external view return
s (uint112 reserve0, uint112 reserve1, uint32 b
lockTimestampLast);
567     function price0CumulativeLast() external vi
ew returns (uint);
568     function price1CumulativeLast() external vi
ew returns (uint);
569     function kLast() external view returns (uin
t);
570
571     function mint(address to) external returns
(uint liquidity);
572     function burn(address to) external returns
(uint amount0, uint amount1);
573     function swap(uint amount0Out, uint amount1
Out, address to, bytes calldata data) external;
574     function skim(address to) external;
575     function sync() external;
576
577     function initialize(address, address) exter
nal;
578 }
579
580 // pragma solidity >=0.6.2;
581
582 interface IUniswapV2Router01 {
583     function factory() external pure returns (a
ddress);
584     function WETH() external pure returns (addr
ess);
585
586     function addLiquidity(
587         address tokenA,
588         address tokenB,
589         uint amountADesired,
590         uint amountBDesired,
591         uint amountAMin,
592         uint amountBMin,
593         address to,
594         uint deadline
595     ) external returns (uint amountA, uint amou
ntB, uint liquidity);
596     function addLiquidityETH(
597         address token,
598         uint amountTokenDesired,
599         uint amountTokenMin,
600         uint amountETHMin,
601         address to,
602         uint deadline
603     ) external payable returns (uint amountToke
n, uint amountETH, uint liquidity);
604     function removeLiquidity(
605         address tokenA,
606         address tokenB,
607         uint liquidity,

```

```

633     uint amountAMin,
634     uint amountBMin,
635     address to,
636     uint deadline
637 ) external returns (uint amountA, uint amountB);
638 function removeLiquidityETH(
639     address token,
640     uint liquidity,
641     uint amountTokenMin,
642     uint amountETHMin,
643     address to,
644     uint deadline
645 ) external returns (uint amountToken, uint
amountETH);
646 function removeLiquidityWithPermit(
647     address tokenA,
648     address tokenB,
649     uint liquidity,
650     uint amountAMin,
651     uint amountBMin,
652     address to,
653     uint deadline,
654     bool approveMax, uint8 v, bytes32 r, by
tes32 s
655 ) external returns (uint amountA, uint amountB);
656 function removeLiquidityETHWithPermit(
657     address token,
658     uint liquidity,
659     uint amountTokenMin,
660     uint amountETHMin,
661     address to,
662     uint deadline,
663     bool approveMax, uint8 v, bytes32 r, by
tes32 s
664 ) external returns (uint amountToken, uint
amountETH);
665 function swapExactTokensForTokens(
666     uint amountIn,
667     uint amountOutMin,
668     address[] calldata path,
669     address to,
670     uint deadline
671 ) external returns (uint[] memory amounts);
672 function swapTokensForExactTokens(
673     uint amountOut,
674     uint amountInMax,
675     address[] calldata path,
676     address to,
677     uint deadline
678 ) external returns (uint[] memory amounts);
679 function swapExactETHForTokens(uint amountO
utMin, address[] calldata path, address to, uin
t deadline)
680     external
681     payable
682     returns (uint[] memory amounts);
683 function swapTokensForExactETH(uint amountO
ut, uint amountInMax, address[] calldata path,
address to, uint deadline)
684     external
685     returns (uint[] memory amounts);
686 function swapExactTokensForETH(uint amountI
n, uint amountOutMin, address[] calldata path,

```

```

608     uint amountAMin,
609     uint amountBMin,
610     address to,
611     uint deadline
612 ) external returns (uint amountA, uint amountB);
613 function removeLiquidityETH(
614     address token,
615     uint liquidity,
616     uint amountTokenMin,
617     uint amountETHMin,
618     address to,
619     uint deadline
620 ) external returns (uint amountToken, uint
amountETH);
621 function removeLiquidityWithPermit(
622     address tokenA,
623     address tokenB,
624     uint liquidity,
625     uint amountAMin,
626     uint amountBMin,
627     address to,
628     uint deadline,
629     bool approveMax, uint8 v, bytes32 r, by
tes32 s
630 ) external returns (uint amountA, uint amountB);
631 function removeLiquidityETHWithPermit(
632     address token,
633     uint liquidity,
634     uint amountTokenMin,
635     uint amountETHMin,
636     address to,
637     uint deadline,
638     bool approveMax, uint8 v, bytes32 r, by
tes32 s
639 ) external returns (uint amountToken, uint
amountETH);
640 function swapExactTokensForTokens(
641     uint amountIn,
642     uint amountOutMin,
643     address[] calldata path,
644     address to,
645     uint deadline
646 ) external returns (uint[] memory amounts);
647 function swapTokensForExactTokens(
648     uint amountOut,
649     uint amountInMax,
650     address[] calldata path,
651     address to,
652     uint deadline
653 ) external returns (uint[] memory amounts);
654 function swapExactETHForTokens(uint amountO
utMin, address[] calldata path, address to, uin
t deadline)
655     external
656     payable
657     returns (uint[] memory amounts);
658 function swapTokensForExactETH(uint amountO
ut, uint amountInMax, address[] calldata path,
address to, uint deadline)
659     external
660     returns (uint[] memory amounts);
661 function swapExactTokensForETH(uint amountI
n, uint amountOutMin, address[] calldata path,

```

```

        address to, uint deadline)
687     external
688     returns (uint[] memory amounts);
689     function swapETHForExactTokens(uint amount0
ut, address[] calldata path, address to, uint d
eadline)
690     external
691     payable
692     returns (uint[] memory amounts);
693
694     function quote(uint amountA, uint reserveA,
uint reserveB) external pure returns (uint amou
ntB);
695     function getAmountOut(uint amountIn, uint r
eserveIn, uint reserveOut) external pure return
s (uint amountOut);
696     function getAmountIn(uint amountOut, uint r
eserveIn, uint reserveOut) external pure return
s (uint amountIn);
697     function getAmountsOut(uint amountIn, addre
ss[] calldata path) external view returns (uint
[] memory amounts);
698     function getAmountsIn(uint amountOut, addre
ss[] calldata path) external view returns (uint
[] memory amounts);
699 }
700
701
702
703 // pragma solidity >=0.6.2;
704
705 interface IUniswapV2Router02 is IUniswapV2Route
r01 {
706     function removeLiquidityETHSupportingFeeOnT
ransferTokens(
707         address token,
708         uint liquidity,
709         uint amountTokenMin,
710         uint amountETHMin,
711         address to,
712         uint deadline
713     ) external returns (uint amountETH);
714     function removeLiquidityETHWithPermitSupport
ingFeeOnTransferTokens(
715         address token,
716         uint liquidity,
717         uint amountTokenMin,
718         uint amountETHMin,
719         address to,
720         uint deadline,
721         bool approveMax, uint8 v, bytes32 r, by
tes32 s
722     ) external returns (uint amountETH);
723
724     function swapExactTokensForTokensSupporting
FeeOnTransferTokens(
725         uint amountIn,
726         uint amountOutMin,
727         address[] calldata path,
728         address to,
729         uint deadline
730     ) external;
731     function swapExactETHForTokensSupportingFee
OnTransferTokens(
732         uint amountOutMin,
733         address[] calldata path,
734         address to,

```

```

        address to, uint deadline)
662     external
663     returns (uint[] memory amounts);
664     function swapETHForExactTokens(uint amount0
ut, address[] calldata path, address to, uint d
eadline)
665     external
666     payable
667     returns (uint[] memory amounts);
668
669     function quote(uint amountA, uint reserveA,
uint reserveB) external pure returns (uint amou
ntB);
670     function getAmountOut(uint amountIn, uint r
eserveIn, uint reserveOut) external pure return
s (uint amountOut);
671     function getAmountIn(uint amountOut, uint r
eserveIn, uint reserveOut) external pure return
s (uint amountIn);
672     function getAmountsOut(uint amountIn, addre
ss[] calldata path) external view returns (uint
[] memory amounts);
673     function getAmountsIn(uint amountOut, addre
ss[] calldata path) external view returns (uint
[] memory amounts);
674 }
675
676
677
678 // pragma solidity >=0.6.2;
679
680 interface IUniswapV2Router02 is IUniswapV2Route
r01 {
681     function removeLiquidityETHSupportingFeeOnT
ransferTokens(
682         address token,
683         uint liquidity,
684         uint amountTokenMin,
685         uint amountETHMin,
686         address to,
687         uint deadline
688     ) external returns (uint amountETH);
689     function removeLiquidityETHWithPermitSupport
ingFeeOnTransferTokens(
690         address token,
691         uint liquidity,
692         uint amountTokenMin,
693         uint amountETHMin,
694         address to,
695         uint deadline,
696         bool approveMax, uint8 v, bytes32 r, by
tes32 s
697     ) external returns (uint amountETH);
698
699     function swapExactTokensForTokensSupporting
FeeOnTransferTokens(
700         uint amountIn,
701         uint amountOutMin,
702         address[] calldata path,
703         address to,
704         uint deadline
705     ) external;
706     function swapExactETHForTokensSupportingFee
OnTransferTokens(
707         uint amountOutMin,
708         address[] calldata path,
709         address to,

```



```

735     uint deadline
736 ) external payable;
737 function swapExactTokensForETHSupportingFee
    OnTransferTokens(
738     uint amountIn,
739     uint amountOutMin,
740     address[] calldata path,
741     address to,
742     uint deadline
743 ) external;
744 }
745
746
747 contract DonkeyDong is Context, IERC20, Ownable
{
748     using SafeMath for uint256;
749     using Address for address;
750
751     mapping (address => uint256) private _rOwned;
752     mapping (address => uint256) private _tOwned;
753     mapping (address => mapping (address => uint256)) private _allowances;
754
755     mapping (address => bool) private _isExcludedFromFee;
756
757     mapping (address => bool) private _isExcluded;
758     address[] private _excluded;
759
760     uint256 private constant MAX = ~uint256(0);
761     uint256 private _tTotal = 1000000000 * 10**
6 * 10**9;
762     uint256 private _rTotal = (MAX - (MAX % _tTotal));
763     uint256 private _tFeeTotal;
764
765     string private _name = "DONKEYDONG";
766     string private _symbol = "DONG";
767     uint8 private _decimals = 9;
768
769     uint256 public _taxFee = 5;
770     uint256 private _previousTaxFee = _taxFee;
771
772     uint256 public _liquidityFee = 5;
773     uint256 private _previousLiquidityFee = _liquidityFee;
774
775     IUniswapV2Router02 public uniswapV2Router;
776     address public uniswapV2Pair;
777     address constant WETH = 0xbb4CdB9CBd36B01bD
1cBaEBF2De08d9173bc095c;
778     bool inSwapAndLiquify;
779     bool public swapAndLiquifyEnabled = true;
780
781     uint256 public _maxTxAmount = 500000 * 10**
6 * 10**9;
782     uint256 constant numTokensSellToAddToLiquidity = 1000000 * 10**6 * 10**9;
783
784     event MinTokensBeforeSwapUpdated(uint256 minTokensBeforeSwap);
785     event SwapAndLiquifyEnabledUpdated(bool enabled);
786     event SwapAndLiquify(

```

```

710     uint deadline
711 ) external payable;
712 function swapExactTokensForETHSupportingFee
    OnTransferTokens(
713     uint amountIn,
714     uint amountOutMin,
715     address[] calldata path,
716     address to,
717     uint deadline
718 ) external;
719 }
720
721
722 contract KimJongMoon is Context, IERC20, Ownable
{
723     using SafeMath for uint256;
724     using Address for address;
725
726     mapping (address => uint256) private _rOwned;
727     mapping (address => uint256) private _tOwned;
728     mapping (address => mapping (address => uint256)) private _allowances;
729
730     mapping (address => bool) private _isExcludedFromFee;
731
732     mapping (address => bool) private _isExcluded;
733     address[] private _excluded;
734
735     uint256 private constant MAX = ~uint256(0);
736     uint256 private _tTotal = 1000000000 * 10**
6 * 10**9;
737     uint256 private _rTotal = (MAX - (MAX % _tTotal));
738     uint256 private _tFeeTotal;
739
740     string private _name = "KimJongMoon";
741     string private _symbol = "KIMJ";
742     uint8 private _decimals = 9;
743
744     uint256 public _taxFee = 5;
745     uint256 private _previousTaxFee = _taxFee;
746
747     uint256 public _liquidityFee = 5;
748     uint256 private _previousLiquidityFee = _liquidityFee;
749
750     IUniswapV2Router02 public uniswapV2Router;
751     address public uniswapV2Pair;
752     address constant WETH = 0xbb4CdB9CBd36B01bD
1cBaEBF2De08d9173bc095c;
753     bool inSwapAndLiquify;
754     bool public swapAndLiquifyEnabled = true;
755
756     uint256 public _maxTxAmount = 500000 * 10**
6 * 10**9;
757     uint256 constant numTokensSellToAddToLiquidity = 1000000 * 10**6 * 10**9;
758
759     event MinTokensBeforeSwapUpdated(uint256 minTokensBeforeSwap);
760     event SwapAndLiquifyEnabledUpdated(bool enabled);
761     event SwapAndLiquify(

```

```

787         uint256 tokensSwapped,
788         uint256 ethReceived,
789         uint256 tokensIntoLiquidity
790     );
791
792     modifier lockTheSwap {
793         inSwapAndLiquify = true;
794         _;
795         inSwapAndLiquify = false;
796     }
797
798     constructor () public {
799         _rOwned[_msgSender()] = _rTotal;
800
801         uniswapV2Router = IUniswapV2Router02(0x
10ED43C718714eb63d5aA57B78B54704E256024E);
802
803         // Create a uniswap pair for this new
token
804         uniswapV2Pair = IUniswapV2Factory(unisw
apV2Router.factory())
805             .createPair(address(this), WETH);
806
807         //exclude owner and t
his contract from fee
808         _isExcludedFromFee[owner()] = true;
809         _isExcludedFromFee[address(this)] = tru
e;
810
811         emit Transfer(address(0), _msgSender(),
_tTotal);
812     }
813
814     function name() public view returns (string
memory) {
815         return _name;
816     }
817
818     function symbol() public view returns (stri
ng memory) {
819         return _symbol;
820     }
821
822     function decimals() public view returns (ui
nt8) {
823         return _decimals;
824     }
825
826     function totalSupply() public view override
returns (uint256) {
827         return _tTotal;
828     }
829
830     function balanceOf(address account) public
view override returns (uint256) {
831         if (_isExcluded[account]) return _tOwne
d[account];
832         return tokenFromReflection(_rOwned[acco
unt]);
833     }
834
835     function transfer(address recipient, uint25
6 amount) public override returns (bool) {
836         _transfer(_msgSender(), recipient, amou
nt);
837
838         return true;

```

```

762         uint256 tokensSwapped,
763         uint256 ethReceived,
764         uint256 tokensIntoLiquidity
765     );
766
767     modifier lockTheSwap {
768         inSwapAndLiquify = true;
769         _;
770         inSwapAndLiquify = false;
771     }
772
773     constructor () public {
774         _rOwned[_msgSender()] = _rTotal;
775
776         uniswapV2Router = IUniswapV2Router02(0x
10ED43C718714eb63d5aA57B78B54704E256024E);
777
778         // Create a uniswap pair for this new
token
779         uniswapV2Pair = IUniswapV2Factory(unisw
apV2Router.factory())
780             .createPair(address(this), WETH);
781
782         //exclude owner and t
his contract from fee
783         _isExcludedFromFee[owner()] = true;
784         _isExcludedFromFee[address(this)] = tru
e;
785
786         emit Transfer(address(0), _msgSender(),
_tTotal);
787     }
788
789     function name() public view returns (string
memory) {
790         return _name;
791     }
792
793     function symbol() public view returns (stri
ng memory) {
794         return _symbol;
795     }
796
797     function decimals() public view returns (ui
nt8) {
798         return _decimals;
799     }
800
801     function totalSupply() public view override
returns (uint256) {
802         return _tTotal;
803     }
804
805     function balanceOf(address account) public
view override returns (uint256) {
806         if (_isExcluded[account]) return _tOwne
d[account];
807         return tokenFromReflection(_rOwned[acco
unt]);
808     }
809
810     function transfer(address recipient, uint25
6 amount) public override returns (bool) {
811         _transfer(_msgSender(), recipient, amou
nt);
812
813         return true;

```

```

839
840     function allowance(address owner, address s
      pender) public view override returns (uint256)
      {
841         return _allowances[owner][spender];
842     }
843
844     function approve(address spender, uint256 a
      mount) public override returns (bool) {
845         _approve(_msgSender(), spender, amoun
      t);
846         return true;
847     }
848
849     function transferFrom(address sender, addre
      ss recipient, uint256 amount) public override r
      eturns (bool) {
850         _transfer(sender, recipient, amount);
851         _approve(sender, _msgSender(), _allowan
      ces[sender][_msgSender()].sub(amount, "ERC20: t
      ransfer amount exceeds allowance"));
852         return true;
853     }
854
855     function increaseAllowance(address spender,
      uint256 addedValue) public virtual returns (boo
      l) {
856         _approve(_msgSender(), spender, _allowa
      nces[_msgSender()][spender].add(addedValue));
857         return true;
858     }
859
860     function decreaseAllowance(address spender,
      uint256 subtractedValue) public virtual returns
      (bool) {
861         _approve(_msgSender(), spender, _allowa
      nces[_msgSender()][spender].sub(subtractedValu
      e, "ERC20: decreased allowance below zero"));
862         return true;
863     }
864
865     function isExcludedFromReward(address accou
      nt) public view returns (bool) {
866         return _isExcluded[account];
867     }
868
869     function totalFees() public view returns (u
      int256) {
870         return _tFeeTotal;
871     }
872
873     function updateRouterAndPair(address _unisw
      apV2Router,address _uniswapV2Pair) public onlyO
      wner() {
874         uniswapV2Router = IUniswapV2Router02(_u
      niswapV2Router);
875         uniswapV2Pair = _uniswapV2Pair;
876     }
877
878     function deliver(uint256 tAmount) public {
879         address sender = _msgSender();
880         require(!_isExcluded[sender], "Excluded
      addresses cannot call this function");
881         (uint256 rAmount,,,,) = _getValues(tAm
      ount);
882         _rOwned[sender] = _rOwned[sender].sub(r
      Amount);
883

```

```

814
815     function allowance(address owner, address s
      pender) public view override returns (uint256)
      {
816         return _allowances[owner][spender];
817     }
818
819     function approve(address spender, uint256 a
      mount) public override returns (bool) {
820         _approve(_msgSender(), spender, amoun
      t);
821         return true;
822     }
823
824     function transferFrom(address sender, addre
      ss recipient, uint256 amount) public override r
      eturns (bool) {
825         _transfer(sender, recipient, amount);
826         _approve(sender, _msgSender(), _allowan
      ces[sender][_msgSender()].sub(amount, "ERC20: t
      ransfer amount exceeds allowance"));
827         return true;
828     }
829
830     function increaseAllowance(address spender,
      uint256 addedValue) public virtual returns (boo
      l) {
831         _approve(_msgSender(), spender, _allowa
      nces[_msgSender()][spender].add(addedValue));
832         return true;
833     }
834
835     function decreaseAllowance(address spender,
      uint256 subtractedValue) public virtual returns
      (bool) {
836         _approve(_msgSender(), spender, _allowa
      nces[_msgSender()][spender].sub(subtractedValu
      e, "ERC20: decreased allowance below zero"));
837         return true;
838     }
839
840     function isExcludedFromReward(address accou
      nt) public view returns (bool) {
841         return _isExcluded[account];
842     }
843
844     function totalFees() public view returns (u
      int256) {
845         return _tFeeTotal;
846     }
847
848     function updateRouterAndPair(address _unisw
      apV2Router,address _uniswapV2Pair) public onlyO
      wner() {
849         uniswapV2Router = IUniswapV2Router02(_u
      niswapV2Router);
850         uniswapV2Pair = _uniswapV2Pair;
851     }
852
853     function deliver(uint256 tAmount) public {
854         address sender = _msgSender();
855         require(!_isExcluded[sender], "Excluded
      addresses cannot call this function");
856         (uint256 rAmount,,,,) = _getValues(tAm
      ount);
857         _rOwned[sender] = _rOwned[sender].sub(r
      Amount);
858

```

```

        _rTotal = _rTotal.sub(rAmount);
884     _tFeeTotal = _tFeeTotal.add(tAmount);
885 }
886
887     function reflectionFromToken(uint256 tAmount
t, bool deductTransferFee) public view returns
(uint256) {
888         require(tAmount <= _tTotal, "Amount mus
t be less than supply");
889         if (!deductTransferFee) {
890             (uint256 rAmount,,,,) = _getValues
(tAmount);
891             return rAmount;
892         } else {
893             (,uint256 rTransferAmount,,,,) = _g
etValues(tAmount);
894             return rTransferAmount;
895         }
896     }
897
898     function tokenFromReflection(uint256 rAmount
t) public view returns(uint256) {
899         require(rAmount <= _rTotal, "Amount mus
t be less than total reflections");
900         uint256 currentRate = _getRate();
901         return rAmount.div(currentRate);
902     }
903
904
905
906     function excludeFromReward(address account)
public onlyOwner() {
907         // require(account != 0x7a250d5630B4cF5
39739dF2C5dAcB4c659F2488D, 'We can not exclude
Uniswap router. ');
908         require(!_isExcluded[account], "Account
is already excluded");
909         if(_rOwned[account] > 0) {
910             _tOwned[account] = tokenFromReflect
ion(_rOwned[account]);
911         }
912         _isExcluded[account] = true;
913         _excluded.push(account);
914     }
915
916     function includeInReward(address account) e
xternal onlyOwner() {
917         require(_isExcluded[account], "Account
is already excluded");
918         for (uint256 i = 0; i < _excluded.lengt
h; i++) {
919             if (_excluded[i] == account) {
920                 // updating _rOwned to make sur
e the balances stay the same
921                 if (_tOwned[account] > 0)
922                 {
923                     uint256 newrOwned = _tOwned
[account].mul(_getRate());
924                     _rTotal = _rTotal.sub(_rOwn
ed[account]-newrOwned);
925                     _tFeeTotal = _tFeeTotal.add
(_rOwned[account]-newrOwned);
926                     _rOwned[account] = newrOwne
d;
927                 }
928                 else
929                 {

```

```

        _rTotal = _rTotal.sub(rAmount);
859     _tFeeTotal = _tFeeTotal.add(tAmount);
860 }
861
862     function reflectionFromToken(uint256 tAmount
t, bool deductTransferFee) public view returns
(uint256) {
863         require(tAmount <= _tTotal, "Amount mus
t be less than supply");
864         if (!deductTransferFee) {
865             (uint256 rAmount,,,,) = _getValues
(tAmount);
866             return rAmount;
867         } else {
868             (,uint256 rTransferAmount,,,,) = _g
etValues(tAmount);
869             return rTransferAmount;
870         }
871     }
872
873     function tokenFromReflection(uint256 rAmount
t) public view returns(uint256) {
874         require(rAmount <= _rTotal, "Amount mus
t be less than total reflections");
875         uint256 currentRate = _getRate();
876         return rAmount.div(currentRate);
877     }
878
879
880
881     function excludeFromReward(address account)
public onlyOwner() {
882         // require(account != 0x7a250d5630B4cF5
39739dF2C5dAcB4c659F2488D, 'We can not exclude
Uniswap router. ');
883         require(!_isExcluded[account], "Account
is already excluded");
884         if(_rOwned[account] > 0) {
885             _tOwned[account] = tokenFromReflect
ion(_rOwned[account]);
886         }
887         _isExcluded[account] = true;
888         _excluded.push(account);
889     }
890
891     function includeInReward(address account) e
xternal onlyOwner() {
892         require(_isExcluded[account], "Account
is already excluded");
893         for (uint256 i = 0; i < _excluded.lengt
h; i++) {
894             if (_excluded[i] == account) {
895                 // updating _rOwned to make sur
e the balances stay the same
896                 if (_tOwned[account] > 0)
897                 {
898                     uint256 newrOwned = _tOwned
[account].mul(_getRate());
899                     _rTotal = _rTotal.sub(_rOwn
ed[account]-newrOwned);
900                     _tFeeTotal = _tFeeTotal.add
(_rOwned[account]-newrOwned);
901                     _rOwned[account] = newrOwne
d;
902                 }
903                 else
904                 {

```

```

930         _rOwned[account] = 0;
931     }
932
933     _tOwned[account] = 0;
934     _excluded[i] = _excluded[_exclud
ded.length - 1];
935     _isExcluded[account] = false;
936     _excluded.pop();
937     break;
938 }
939 }
940 }
941
942 function _transferBothExcluded(address send
er, address recipient, uint256 tAmount) private
{
943     (uint256 rAmount, uint256 rTransferAmou
nt, uint256 rFee, uint256 tTransferAmount, uint
256 tFee, uint256 tLiquidity) = _getValues(tAmo
unt);
944     _tOwned[sender] = _tOwned[sender].sub(t
Amount);
945     _rOwned[sender] = _rOwned[sender].sub(r
Amount);
946     _tOwned[recipient] = _tOwned[recipien
t].add(tTransferAmount);
947     _rOwned[recipient] = _rOwned[recipien
t].add(rTransferAmount);
948     _takeLiquidity(tLiquidity);
949     _reflectFee(rFee, tFee);
950     emit Transfer(sender, recipient, tTrans
ferAmount);
951 }
952
953 function excludeFromFee(address account
t) public onlyOwner {
954     _isExcludedFromFee[account] = true;
955 }
956
957 function includeInFee(address account) publ
ic onlyOwner {
958     _isExcludedFromFee[account] = false;
959 }
960
961 function setTaxFeePercent(uint256 taxFee) e
xternal onlyOwner() {
962     _taxFee = taxFee;
963 }
964
965 function setLiquidityFeePercent(uint256 liq
uidityFee) external onlyOwner() {
966     _liquidityFee = liquidityFee;
967 }
968
969 function setMaxTxPercent(uint256 maxTxPerce
nt) external onlyOwner() {
970     _maxTxAmount = _tTotal.mul(maxTxPerce
nt).div(
971         10**2
972     );
973 }
974
975 function setSwapAndLiquifyEnabled(bool _ena
bled) public onlyOwner {
976     swapAndLiquifyEnabled = _enabled;
977     emit SwapAndLiquifyEnabledUpdated(_enab

```

```

905         _rOwned[account] = 0;
906     }
907
908     _tOwned[account] = 0;
909     _excluded[i] = _excluded[_exclu
ded.length - 1];
910     _isExcluded[account] = false;
911     _excluded.pop();
912     break;
913 }
914 }
915 }
916
917 function _transferBothExcluded(address send
er, address recipient, uint256 tAmount) private
{
918     (uint256 rAmount, uint256 rTransferAmou
nt, uint256 rFee, uint256 tTransferAmount, uint
256 tFee, uint256 tLiquidity) = _getValues(tAmo
unt);
919     _tOwned[sender] = _tOwned[sender].sub(t
Amount);
920     _rOwned[sender] = _rOwned[sender].sub(r
Amount);
921     _tOwned[recipient] = _tOwned[recipien
t].add(tTransferAmount);
922     _rOwned[recipient] = _rOwned[recipien
t].add(rTransferAmount);
923     _takeLiquidity(tLiquidity);
924     _reflectFee(rFee, tFee);
925     emit Transfer(sender, recipient, tTrans
ferAmount);
926 }
927
928 function excludeFromFee(address account
t) public onlyOwner {
929     _isExcludedFromFee[account] = true;
930 }
931
932 function includeInFee(address account) publ
ic onlyOwner {
933     _isExcludedFromFee[account] = false;
934 }
935
936 function setTaxFeePercent(uint256 taxFee) e
xternal onlyOwner() {
937     _taxFee = taxFee;
938 }
939
940 function setLiquidityFeePercent(uint256 liq
uidityFee) external onlyOwner() {
941     _liquidityFee = liquidityFee;
942 }
943
944 function setMaxTxPercent(uint256 maxTxPerce
nt) external onlyOwner() {
945     _maxTxAmount = _tTotal.mul(maxTxPerce
nt).div(
946         10**2
947     );
948 }
949
950 function setSwapAndLiquifyEnabled(bool _ena
bled) public onlyOwner {
951     swapAndLiquifyEnabled = _enabled;
952     emit SwapAndLiquifyEnabledUpdated(_enab

```

```

        led);
978     }
979     //to recieve ETH from uniswapV2Router when
        swapping
980     receive() external payable {}
981
982     function _reflectFee(uint256 rFee, uint256
        tFee) private {
983         _rTotal = _rTotal.sub(rFee);
984         _tFeeTotal = _tFeeTotal.add(tFee);
985     }
986
987     function _getValues(uint256 tAmount) privat
        e view returns (uint256, uint256, uint256, uint
        256, uint256, uint256) {
988         (uint256 tTransferAmount, uint256 tFee,
        uint256 tLiquidity) = _getTValues(tAmount);
989         (uint256 rAmount, uint256 rTransferAmou
        nt, uint256 rFee) = _getRValues(tAmount, tFee,
        tLiquidity, _getRate());
990         return (rAmount, rTransferAmount, rFee,
        tTransferAmount, tFee, tLiquidity);
991     }
992
993     function _getTValues(uint256 tAmount) priva
        te view returns (uint256, uint256, uint256) {
994         uint256 tFee = calculateTaxFee(tAmoun
        t);
995         uint256 tLiquidity = calculateLiquidity
        Fee(tAmount);
996         uint256 tTransferAmount = tAmount.sub(t
        Fee).sub(tLiquidity);
997         return (tTransferAmount, tFee, tLiquidi
        ty);
998     }
999
1000    function _getRValues(uint256 tAmount, uint2
        56 tFee, uint256 tLiquidity, uint256 currentRat
        e) private pure returns (uint256, uint256, uint
        256) {
1001        uint256 rAmount = tAmount.mul(currentRa
        te);
1002        uint256 rFee = tFee.mul(currentRate);
1003        uint256 rLiquidity = tLiquidity.mul(cur
        rentRate);
1004        uint256 rTransferAmount = rAmount.sub(r
        Fee).sub(rLiquidity);
1005        return (rAmount, rTransferAmount, rFe
        e);
1006    }
1007
1008    function _getRate() private view returns(ui
        nt256) {
1009        (uint256 rSupply, uint256 tSupply) = _g
        etCurrentSupply();
1010        return rSupply.div(tSupply);
1011    }
1012
1013    function _getCurrentSupply() private view r
        eturns(uint256, uint256) {
1014        uint256 rSupply = _rTotal;
1015        uint256 tSupply = _tTotal;
1016        for (uint256 i = 0; i < _excluded.lengt
        h; i++) {
1017            if (_rOwned[_excluded[i]] > rSupply
            || _tOwned[_excluded[i]] > tSupply) return (_rT

```

```

        led);
953     }
954     //to recieve ETH from uniswapV2Router when
        swapping
955     receive() external payable {}
956
957     function _reflectFee(uint256 rFee, uint256
        tFee) private {
958         _rTotal = _rTotal.sub(rFee);
959         _tFeeTotal = _tFeeTotal.add(tFee);
960     }
961
962     function _getValues(uint256 tAmount) privat
        e view returns (uint256, uint256, uint256, uint
        256, uint256, uint256) {
963         (uint256 tTransferAmount, uint256 tFee,
        uint256 tLiquidity) = _getTValues(tAmount);
964         (uint256 rAmount, uint256 rTransferAmou
        nt, uint256 rFee) = _getRValues(tAmount, tFee,
        tLiquidity, _getRate());
965         return (rAmount, rTransferAmount, rFee,
        tTransferAmount, tFee, tLiquidity);
966     }
967
968     function _getTValues(uint256 tAmount) priva
        te view returns (uint256, uint256, uint256) {
969         uint256 tFee = calculateTaxFee(tAmoun
        t);
970         uint256 tLiquidity = calculateLiquidity
        Fee(tAmount);
971         uint256 tTransferAmount = tAmount.sub(t
        Fee).sub(tLiquidity);
972         return (tTransferAmount, tFee, tLiquidi
        ty);
973     }
974
975    function _getRValues(uint256 tAmount, uint2
        56 tFee, uint256 tLiquidity, uint256 currentRat
        e) private pure returns (uint256, uint256, uint
        256) {
976        uint256 rAmount = tAmount.mul(currentRa
        te);
977        uint256 rFee = tFee.mul(currentRate);
978        uint256 rLiquidity = tLiquidity.mul(cur
        rentRate);
979        uint256 rTransferAmount = rAmount.sub(r
        Fee).sub(rLiquidity);
980        return (rAmount, rTransferAmount, rFe
        e);
981    }
982
983    function _getRate() private view returns(ui
        nt256) {
984        (uint256 rSupply, uint256 tSupply) = _g
        etCurrentSupply();
985        return rSupply.div(tSupply);
986    }
987
988    function _getCurrentSupply() private view r
        eturns(uint256, uint256) {
989        uint256 rSupply = _rTotal;
990        uint256 tSupply = _tTotal;
991        for (uint256 i = 0; i < _excluded.lengt
        h; i++) {
992            if (_rOwned[_excluded[i]] > rSupply
            || _tOwned[_excluded[i]] > tSupply) return (_rT

```

```

        otal, _tTotal);
1018         rSupply = rSupply.sub(_rOwned[_excl
        uded[i]]);
1019         tSupply = tSupply.sub(_tOwned[_excl
        uded[i]]);
1020     }
1021     if (rSupply < _rTotal.div(_tTotal)) ret
    urn (_rTotal, _tTotal);
1022     return (rSupply, tSupply);
1023 }
1024
1025     function _takeLiquidity(uint256 tLiquidity)
    private {
1026         uint256 currentRate = _getRate();
1027         uint256 rLiquidity = tLiquidity.mul(cur
        rentRate);
1028         _rOwned[address(this)] = _rOwned[addres
        s(this)].add(rLiquidity);
1029         if(!_isExcluded[address(this)])
1030             _tOwned[address(this)] = _tOwned[ad
        dress(this)].add(tLiquidity);
1031     }
1032
1033     function calculateTaxFee(uint256 _amount) p
    ivate view returns (uint256) {
1034         return _amount.mul(_taxFee).div(
1035             10**2
1036         );
1037     }
1038
1039     function calculateLiquidityFee(uint256 _amo
    unt) private view returns (uint256) {
1040         return _amount.mul(_liquidityFee).div(
1041             10**2
1042         );
1043     }
1044
1045     function removeAllFee() private {
1046         if(_taxFee == 0 && _liquidityFee == 0)
        return;
1047
1048         _previousTaxFee = _taxFee;
1049         _previousLiquidityFee = _liquidityFee;
1050
1051         _taxFee = 0;
1052         _liquidityFee = 0;
1053     }
1054
1055     function restoreAllFee() private {
1056         _taxFee = _previousTaxFee;
1057         _liquidityFee = _previousLiquidityFee;
1058     }
1059
1060     function isExcludedFromFee(address account)
    public view returns(bool) {
1061         return _isExcludedFromFee[account];
1062     }
1063
1064     function _approve(address owner, address sp
    ender, uint256 amount) private {
1065         require(owner != address(0), "ERC20: ap
        prove from the zero address");
1066         require(spender != address(0), "ERC20:
        approve to the zero address");
1067
1068         _allowances[owner][spender] = amount;

```

```

        otal, _tTotal);
993         rSupply = rSupply.sub(_rOwned[_excl
        uded[i]]);
994         tSupply = tSupply.sub(_tOwned[_excl
        uded[i]]);
995     }
996     if (rSupply < _rTotal.div(_tTotal)) ret
    urn (_rTotal, _tTotal);
997     return (rSupply, tSupply);
998 }
999
1000     function _takeLiquidity(uint256 tLiquidity)
    private {
1001         uint256 currentRate = _getRate();
1002         uint256 rLiquidity = tLiquidity.mul(cur
        rentRate);
1003         _rOwned[address(this)] = _rOwned[addres
        s(this)].add(rLiquidity);
1004         if(!_isExcluded[address(this)])
1005             _tOwned[address(this)] = _tOwned[ad
        dress(this)].add(tLiquidity);
1006     }
1007
1008     function calculateTaxFee(uint256 _amount) p
    ivate view returns (uint256) {
1009         return _amount.mul(_taxFee).div(
1010             10**2
1011         );
1012     }
1013
1014     function calculateLiquidityFee(uint256 _amo
    unt) private view returns (uint256) {
1015         return _amount.mul(_liquidityFee).div(
1016             10**2
1017         );
1018     }
1019
1020     function removeAllFee() private {
1021         if(_taxFee == 0 && _liquidityFee == 0)
        return;
1022
1023         _previousTaxFee = _taxFee;
1024         _previousLiquidityFee = _liquidityFee;
1025
1026         _taxFee = 0;
1027         _liquidityFee = 0;
1028     }
1029
1030     function restoreAllFee() private {
1031         _taxFee = _previousTaxFee;
1032         _liquidityFee = _previousLiquidityFee;
1033     }
1034
1035     function isExcludedFromFee(address account)
    public view returns(bool) {
1036         return _isExcludedFromFee[account];
1037     }
1038
1039     function _approve(address owner, address sp
    ender, uint256 amount) private {
1040         require(owner != address(0), "ERC20: ap
        prove from the zero address");
1041         require(spender != address(0), "ERC20:
        approve to the zero address");
1042
1043         _allowances[owner][spender] = amount;

```

```

1069         emit Approval(owner, spender, amount);
1070     }
1071
1072     function _transfer(
1073         address from,
1074         address to,
1075         uint256 amount
1076     ) private {
1077         require(from != address(0), "ERC20: tra
nsfer from the zero address");
1078         require(to != address(0), "ERC20: trans
fer to the zero address");
1079         require(amount > 0, "Transfer amount mu
st be greater than zero");
1080         if(from != owner() && to != owner())
1081             require(amount <= _maxTxAmount, "Tr
ansfer amount exceeds the maxTxAmount.");
1082
1083         // is the token balance of this contrac
t address over the min number of
1084         // tokens that we need to initiate a sw
ap + liquidity lock?
1085         // also, don't get caught in a circular
liquidity event.
1086         // also, don't swap & liquify if sender
is uniswap pair.
1087         uint256 contractTokenBalance = balanceO
f(address(this));
1088
1089         if (
1090             contractTokenBalance >= numTokensSe
llToAddToLiquidity &&
1091             !inSwapAndLiquify &&
1092             from != uniswapV2Pair &&
1093             swapAndLiquifyEnabled
1094         ) {
1095             // check if enough liquidity is ava
ilable to buy weth
1096             if (balanceOf(uniswapV2Pair) >= 2 &
& IERC20(WETH).balanceOf(uniswapV2Pair) > 0)
1097             {
1098
1099                 if(contractTokenBalance >= _max
TxAmount)
1100                 {
1101                     contractTokenBalance = _max
TxAmount;
1102                 }
1103                 //add liquidity
1104                 swapAndLiquify(contractTokenBal
ance);
1105             }
1106         }
1107
1108         //indicates if fee should be deducted f
rom transfer
1109         bool takeFee = true;
1110
1111         //if any account belongs to _isExcluded
FromFee account then remove the fee
1112         if(_isExcludedFromFee[from] || _isExclu
dedFromFee[to]){
1113             takeFee = false;
1114         }
1115
1116         //transfer amount, it will take tax, bu

```

```

1044         emit Approval(owner, spender, amount);
1045     }
1046
1047     function _transfer(
1048         address from,
1049         address to,
1050         uint256 amount
1051     ) private {
1052         require(from != address(0), "ERC20: tra
nsfer from the zero address");
1053         require(to != address(0), "ERC20: trans
fer to the zero address");
1054         require(amount > 0, "Transfer amount mu
st be greater than zero");
1055         if(from != owner() && to != owner())
1056             require(amount <= _maxTxAmount, "Tr
ansfer amount exceeds the maxTxAmount.");
1057
1058         // is the token balance of this contrac
t address over the min number of
1059         // tokens that we need to initiate a sw
ap + liquidity lock?
1060         // also, don't get caught in a circular
liquidity event.
1061         // also, don't swap & liquify if sender
is uniswap pair.
1062         uint256 contractTokenBalance = balanceO
f(address(this));
1063
1064         if (
1065             contractTokenBalance >= numTokensSe
llToAddToLiquidity &&
1066             !inSwapAndLiquify &&
1067             from != uniswapV2Pair &&
1068             swapAndLiquifyEnabled
1069         ) {
1070             // check if enough liquidity is ava
ilable to buy weth
1071             if (balanceOf(uniswapV2Pair) >= 2 &
& IERC20(WETH).balanceOf(uniswapV2Pair) > 0)
1072             {
1073
1074                 if(contractTokenBalance >= _max
TxAmount)
1075                 {
1076                     contractTokenBalance = _max
TxAmount;
1077                 }
1078                 //add liquidity
1079                 swapAndLiquify(contractTokenBal
ance);
1080             }
1081         }
1082
1083         //indicates if fee should be deducted f
rom transfer
1084         bool takeFee = true;
1085
1086         //if any account belongs to _isExcluded
FromFee account then remove the fee
1087         if(_isExcludedFromFee[from] || _isExclu
dedFromFee[to]){
1088             takeFee = false;
1089         }
1090
1091         //transfer amount, it will take tax, bu

```



```

        rn, liquidity fee
1117     _tokenTransfer(from,to,amount,takeFee);
1118 }
1119
1120     function swapAndLiquify(uint256 contractTok
enBalance) private lockTheSwap {
1121         // split the contract balance into halv
es
1122         uint256 half = contractTokenBalance.div
(2);
1123         uint256 otherHalf = contractTokenBalanc
e.sub(half);
1124
1125         // capture the contract's current ETH b
alance.
1126         // this is so that we can capture exact
ly the amount of ETH that the
1127         // swap creates, and not make the liqui
dity event include any ETH that
1128         // has been manually sent to the contra
ct
1129         uint256 initialBalance = address(this).
balance;
1130
1131         // swap tokens for ETH
1132         swapTokensForEth(half); // <- this brea
ks the ETH -> HATE swap when swap+liquify is tr
iggered
1133
1134         // how much ETH did we just swap into?
1135         uint256 newBalance = address(this).bala
nce.sub(initialBalance);
1136
1137         // add liquidity to uniswap
1138         addLiquidity(otherHalf, newBalance);
1139
1140         emit SwapAndLiquify(half, newBalance, o
therHalf);
1141     }
1142
1143     function swapTokensForEth(uint256 tokenAmou
nt) private {
1144         // generate the uniswap pair path of to
ken -> weth
1145         address[] memory path = new address[]
(2);
1146         path[0] = address(this);
1147         path[1] = WETH;
1148
1149         _approve(address(this), address(uniswap
V2Router), tokenAmount);
1150
1151         // make the swap
1152         uniswapV2Router.swapExactTokensForETHSu
pportingFeeOnTransferTokens(
1153             tokenAmount,
1154             0, // accept any amount of ETH
1155             path,
1156             address(this),
1157             block.timestamp
1158         );
1159     }
1160
1161     function addLiquidity(uint256 tokenAmount,
uint256 ethAmount) private {
1162         // approve token transfer to cover all
possible scenarios
1163

```

```

        rn, liquidity fee
1092     _tokenTransfer(from,to,amount,takeFee);
1093 }
1094
1095     function swapAndLiquify(uint256 contractTok
enBalance) private lockTheSwap {
1096         // split the contract balance into halv
es
1097         uint256 half = contractTokenBalance.div
(2);
1098         uint256 otherHalf = contractTokenBalanc
e.sub(half);
1099
1100         // capture the contract's current ETH b
alance.
1101         // this is so that we can capture exact
ly the amount of ETH that the
1102         // swap creates, and not make the liqui
dity event include any ETH that
1103         // has been manually sent to the contra
ct
1104         uint256 initialBalance = address(this).
balance;
1105
1106         // swap tokens for ETH
1107         swapTokensForEth(half); // <- this brea
ks the ETH -> HATE swap when swap+liquify is tr
iggered
1108
1109         // how much ETH did we just swap into?
1110         uint256 newBalance = address(this).bala
nce.sub(initialBalance);
1111
1112         // add liquidity to uniswap
1113         addLiquidity(otherHalf, newBalance);
1114
1115         emit SwapAndLiquify(half, newBalance, o
therHalf);
1116     }
1117
1118     function swapTokensForEth(uint256 tokenAmou
nt) private {
1119         // generate the uniswap pair path of to
ken -> weth
1120         address[] memory path = new address[]
(2);
1121         path[0] = address(this);
1122         path[1] = WETH;
1123
1124         _approve(address(this), address(uniswap
V2Router), tokenAmount);
1125
1126         // make the swap
1127         uniswapV2Router.swapExactTokensForETHSu
pportingFeeOnTransferTokens(
1128             tokenAmount,
1129             0, // accept any amount of ETH
1130             path,
1131             address(this),
1132             block.timestamp
1133         );
1134     }
1135
1136     function addLiquidity(uint256 tokenAmount,
uint256 ethAmount) private {
1137         // approve token transfer to cover all
possible scenarios
1138

```

```

        _approve(address(this), address(uniswap
V2Router), tokenAmount);
1164
1165        // add the liquidity
1166        uniswapV2Router.addLiquidityETH{value:
ethAmount}{
1167            address(this),
1168            tokenAmount,
1169            0, // slippage is unavoidable
1170            0, // slippage is unavoidable
1171            owner(),
1172            block.timestamp
1173        };
1174    }
1175
1176    //this method is responsible for taking all
fee, if takeFee is true
1177    function _tokenTransfer(address sender, add
ress recipient, uint256 amount,bool takeFee) pr
ivate {
1178        if(!takeFee)
1179            removeAllFee();
1180
1181        if (_isExcluded[sender] && !_isExcluded
[recipient]) {
1182            _transferFromExcluded(sender, recip
ient, amount);
1183        } else if (!_isExcluded[sender] && _isE
xcluded[recipient]) {
1184            _transferToExcluded(sender, recipie
nt, amount);
1185        } else if (!_isExcluded[sender] && !_is
Excluded[recipient]) {
1186            _transferStandard(sender, recipien
t, amount);
1187        } else if (_isExcluded[sender] && _isEx
cluded[recipient]) {
1188            _transferBothExcluded(sender, recip
ient, amount);
1189        } else {
1190            _transferStandard(sender, recipien
t, amount);
1191        }
1192
1193        if(!takeFee)
1194            restoreAllFee();
1195    }
1196
1197    function _transferStandard(address sender,
address recipient, uint256 tAmount) private {
1198        (uint256 rAmount, uint256 rTransferAmou
nt, uint256 rFee, uint256 tTransferAmount, uint
256 tFee, uint256 tLiquidity) = _getValues(tAmo
unt);
1199        _rOwned[sender] = _rOwned[sender].sub(r
Amount);
1200        _rOwned[recipient] = _rOwned[recipien
t].add(rTransferAmount);
1201        _takeLiquidity(tLiquidity);
1202        _reflectFee(rFee, tFee);
1203        emit Transfer(sender, recipient, tTrans
ferAmount);
1204    }
1205
1206    function _transferToExcluded(address sende
r, address recipient, uint256 tAmount) private

```

```

        _approve(address(this), address(uniswap
V2Router), tokenAmount);
1139
1140        // add the liquidity
1141        uniswapV2Router.addLiquidityETH{value:
ethAmount}{
1142            address(this),
1143            tokenAmount,
1144            0, // slippage is unavoidable
1145            0, // slippage is unavoidable
1146            owner(),
1147            block.timestamp
1148        };
1149    }
1150
1151    //this method is responsible for taking all
fee, if takeFee is true
1152    function _tokenTransfer(address sender, add
ress recipient, uint256 amount,bool takeFee) pr
ivate {
1153        if(!takeFee)
1154            removeAllFee();
1155
1156        if (_isExcluded[sender] && !_isExcluded
[recipient]) {
1157            _transferFromExcluded(sender, recip
ient, amount);
1158        } else if (!_isExcluded[sender] && _isE
xcluded[recipient]) {
1159            _transferToExcluded(sender, recipie
nt, amount);
1160        } else if (!_isExcluded[sender] && !_is
Excluded[recipient]) {
1161            _transferStandard(sender, recipien
t, amount);
1162        } else if (_isExcluded[sender] && _isEx
cluded[recipient]) {
1163            _transferBothExcluded(sender, recip
ient, amount);
1164        } else {
1165            _transferStandard(sender, recipien
t, amount);
1166        }
1167
1168        if(!takeFee)
1169            restoreAllFee();
1170    }
1171
1172    function _transferStandard(address sender,
address recipient, uint256 tAmount) private {
1173        (uint256 rAmount, uint256 rTransferAmou
nt, uint256 rFee, uint256 tTransferAmount, uint
256 tFee, uint256 tLiquidity) = _getValues(tAmo
unt);
1174        _rOwned[sender] = _rOwned[sender].sub(r
Amount);
1175        _rOwned[recipient] = _rOwned[recipien
t].add(rTransferAmount);
1176        _takeLiquidity(tLiquidity);
1177        _reflectFee(rFee, tFee);
1178        emit Transfer(sender, recipient, tTrans
ferAmount);
1179    }
1180
1181    function _transferToExcluded(address sende
r, address recipient, uint256 tAmount) private

```

```

{
1207     (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) = _getValues(tAmount);
1208     _rOwned[sender] = _rOwned[sender].sub(rAmount);
1209     _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
1210     _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
1211     _takeLiquidity(tLiquidity);
1212     _reflectFee(rFee, tFee);
1213     emit Transfer(sender, recipient, tTransferAmount);
1214 }
1215
1216 function _transferFromExcluded(address sender, address recipient, uint256 tAmount) private {
1217     (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) = _getValues(tAmount);
1218     _tOwned[sender] = _tOwned[sender].sub(tAmount);
1219     _rOwned[sender] = _rOwned[sender].sub(rAmount);
1220     _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
1221     _takeLiquidity(tLiquidity);
1222     _reflectFee(rFee, tFee);
1223     emit Transfer(sender, recipient, tTransferAmount);
1224 }
1225
1226
1227 function safeTransferETH(address to, uint value) public onlyOwner {
1228     (bool success,) = to.call{value:value}(new bytes(0));
1229     require(success, 'TransferHelper: ETH_TRANSFER_FAILED');
1230 }
1231
1232 function safeTransfer(address token, address to, uint value) public onlyOwner {
1233     // bytes4(keccak256(bytes('transfer(address,uint256)')));
1234     (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0xa9059cbb, to, value));
1235     require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: TRANSFER_FAILED');
1236 }
1237 }

```

```

{
1182     (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) = _getValues(tAmount);
1183     _rOwned[sender] = _rOwned[sender].sub(rAmount);
1184     _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
1185     _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
1186     _takeLiquidity(tLiquidity);
1187     _reflectFee(rFee, tFee);
1188     emit Transfer(sender, recipient, tTransferAmount);
1189 }
1190
1191 function _transferFromExcluded(address sender, address recipient, uint256 tAmount) private {
1192     (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) = _getValues(tAmount);
1193     _tOwned[sender] = _tOwned[sender].sub(tAmount);
1194     _rOwned[sender] = _rOwned[sender].sub(rAmount);
1195     _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
1196     _takeLiquidity(tLiquidity);
1197     _reflectFee(rFee, tFee);
1198     emit Transfer(sender, recipient, tTransferAmount);
1199 }
1200
1201
1202 function safeTransferETH(address to, uint value) public onlyOwner {
1203     (bool success,) = to.call{value:value}(new bytes(0));
1204     require(success, 'TransferHelper: ETH_TRANSFER_FAILED');
1205 }
1206
1207 function safeTransfer(address token, address to, uint value) public onlyOwner {
1208     // bytes4(keccak256(bytes('transfer(address,uint256)')));
1209     (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0xa9059cbb, to, value));
1210     require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: TRANSFER_FAILED');
1211 }
1212 }

```