```solidity
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.8.0) (token/ERC20/ERC20.sol)

pragma solidity ^0.8.0;

import "./IERC20.sol";
import "./extensions/IERC20Metadata.sol";
import "../../utils/Context.sol";

/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using {_mint}.
 * For a generic mechanism see {ERC20PresetMinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.openzeppelin.com/t/how-to-implement-erc20-supply-mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin Contracts guidelines: functions revert
 * instead returning `false` on failure. This behavior is nonetheless
 * conventional and does not conflict with the expectations of ERC20
 * applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See {IERC20-approve}.
 */
contract ERC20 is Context, IERC20, IERC20Metadata {
    mapping(address => uint256) private _balances;

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

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;

    /**
     * @dev Sets the values for {name} and {symbol}.
     *
     * The default value of {decimals} is 18. To select a different value for
     * {decimals} you should overload it.
     *
     * All two of these values are immutable: they can only be set once during
     * construction.
     */
    constructor(string memory name_, string memory symbol_) {
        _name = name_;
        _symbol = symbol_;
    }

    /**
     * @dev Returns the name of the token.
     */
    function name() public view virtual override returns (string memory) {
        return _name;
    }

    /**
     * @dev Returns the symbol of the token, usually a shorter version of the
     * name.
     */
    function symbol() public view virtual override returns (string memory) {
        return _symbol;
    }

```

```
74        /**
75         * @dev Returns the number of decimals used to get its user representation.
76         * For example, if `decimals` equals `2`, a balance of `505` tokens should
77         * be displayed to a user as `5.05` (`505 / 10 ** 2`).
78         *
79         * Tokens usually opt for a value of 18, imitating the relationship between
80         * Ether and Wei. This is the value {ERC20} uses, unless this function is
81         * overridden;
82         *
83         * NOTE: This information is only used for _display_ purposes: it in
84         * no way affects any of the arithmetic of the contract, including
85         * {IERC20-balanceOf} and {IERC20-transfer}.
86         */
87        function decimals() public view virtual override returns (uint8) {
88            return 18;
89        }
90
91        /**
92         * @dev See {IERC20-totalSupply}.
93         */
94        function totalSupply() public view virtual override returns (uint256) {
95            return _totalSupply;
96        }
97
98        /**
99         * @dev See {IERC20-balanceOf}.
100        */
101       function balanceOf(address account) public view virtual override returns (uint256)
          {
102           return _balances[account];
103       }
104
105       /**
106        * @dev See {IERC20-transfer}.
107        *
108        * Requirements:
109        *
110        * - `to` cannot be the zero address.
111        * - the caller must have a balance of at least `amount`.
112        */
113       function transfer(address to, uint256 amount) public virtual override returns (
          bool) {
114           address owner = _msgSender();
115           _transfer(owner, to, amount);
116           return true;
117       }
118
119       /**
120        * @dev See {IERC20-allowance}.
121        */
122       function allowance(address owner, address spender) public view virtual override
          returns (uint256) {
123           return _allowances[owner][spender];
124       }
125
126       /**
127        * @dev See {IERC20-approve}.
128        *
129        * NOTE: If `amount` is the maximum `uint256`, the allowance is not updated on
130        * `transferFrom`. This is semantically equivalent to an infinite approval.
131        *
132        * Requirements:
133        *
134        * - `spender` cannot be the zero address.
135        */
136       function approve(address spender, uint256 amount) public virtual override returns
          (bool) {
137           address owner = _msgSender();
138           _approve(owner, spender, amount);
139           return true;
140       }
141
142       /**
```

```
143         * @dev See {IERC20-transferFrom}.
144         *
145         * Emits an {Approval} event indicating the updated allowance. This is not
146         * required by the EIP. See the note at the beginning of {ERC20}.
147         *
148         * NOTE: Does not update the allowance if the current allowance
149         * is the maximum `uint256`.
150         *
151         * Requirements:
152         *
153         * - `from` and `to` cannot be the zero address.
154         * - `from` must have a balance of at least `amount`.
155         * - the caller must have allowance for ``from``'s tokens of at least
156         * `amount`.
157         */
158        function transferFrom(
159            address from,
160            address to,
161            uint256 amount
162        ) public virtual override returns (bool) {
163            address spender = _msgSender();
164            _spendAllowance(from, spender, amount);
165            _transfer(from, to, amount);
166            return true;
167        }
168
169        /**
170         * @dev Atomically increases the allowance granted to `spender` by the caller.
171         *
172         * This is an alternative to {approve} that can be used as a mitigation for
173         * problems described in {IERC20-approve}.
174         *
175         * Emits an {Approval} event indicating the updated allowance.
176         *
177         * Requirements:
178         *
179         * - `spender` cannot be the zero address.
180         */
181        function increaseAllowance(address spender, uint256 addedValue) public virtual
        returns (bool) {
182            address owner = _msgSender();
183            _approve(owner, spender, allowance(owner, spender) + addedValue);
184            return true;
185        }
186
187        /**
188         * @dev Atomically decreases the allowance granted to `spender` by the caller.
189         *
190         * This is an alternative to {approve} that can be used as a mitigation for
191         * problems described in {IERC20-approve}.
192         *
193         * Emits an {Approval} event indicating the updated allowance.
194         *
195         * Requirements:
196         *
197         * - `spender` cannot be the zero address.
198         * - `spender` must have allowance for the caller of at least
199         * `subtractedValue`.
200         */
201        function decreaseAllowance(address spender, uint256 subtractedValue) public
        virtual returns (bool) {
202            address owner = _msgSender();
203            uint256 currentAllowance = allowance(owner, spender);
204            require(currentAllowance >= subtractedValue, "ERC20: decreased allowance
            below zero");
205            unchecked {
206                _approve(owner, spender, currentAllowance - subtractedValue);
207            }
208
209            return true;
210        }
211
212        /**
```

```
213         * @dev Moves `amount` of tokens from `from` to `to`.
214         *
215         * This internal function is equivalent to {transfer}, and can be used to
216         * e.g. implement automatic token fees, slashing mechanisms, etc.
217         *
218         * Emits a {Transfer} event.
219         *
220         * Requirements:
221         *
222         * - `from` cannot be the zero address.
223         * - `to` cannot be the zero address.
224         * - `from` must have a balance of at least `amount`.
225         */
226        function _transfer(
227            address from,
228            address to,
229            uint256 amount
230        ) internal virtual {
231            require(from != address(0), "ERC20: transfer from the zero address");
232            require(to != address(0), "ERC20: transfer to the zero address");
233
234            _beforeTokenTransfer(from, to, amount);
235
236            uint256 fromBalance = _balances[from];
237            require(fromBalance >= amount, "ERC20: transfer amount exceeds balance");
238            unchecked {
239                _balances[from] = fromBalance - amount;
240                // Overflow not possible: the sum of all balances is capped by
                    totalSupply, and the sum is preserved by
241                // decrementing then incrementing.
242                _balances[to] += amount;
243            }
244
245            emit Transfer(from, to, amount);
246
247            _afterTokenTransfer(from, to, amount);
248        }
249
250        /** @dev Creates `amount` tokens and assigns them to `account`, increasing
251         * the total supply.
252         *
253         * Emits a {Transfer} event with `from` set to the zero address.
254         *
255         * Requirements:
256         *
257         * - `account` cannot be the zero address.
258         */
259        function _mint(address account, uint256 amount) internal virtual {
260            require(account != address(0), "ERC20: mint to the zero address");
261
262            _beforeTokenTransfer(address(0), account, amount);
263
264            _totalSupply += amount;
265            unchecked {
266                // Overflow not possible: balance + amount is at most totalSupply +
                    amount, which is checked above.
267                _balances[account] += amount;
268            }
269            emit Transfer(address(0), account, amount);
270
271            _afterTokenTransfer(address(0), account, amount);
272        }
273
274        /**
275         * @dev Destroys `amount` tokens from `account`, reducing the
276         * total supply.
277         *
278         * Emits a {Transfer} event with `to` set to the zero address.
279         *
280         * Requirements:
281         *
282         * - `account` cannot be the zero address.
283         * - `account` must have at least `amount` tokens.
```

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

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

        uint256 accountBalance = _balances[account];
        require(accountBalance >= amount, "ERC20: burn amount exceeds balance");
        unchecked {
            _balances[account] = accountBalance - amount;
            // Overflow not possible: amount <= accountBalance <= totalSupply.
            _totalSupply -= amount;
        }

        emit Transfer(account, address(0), amount);

        _afterTokenTransfer(account, address(0), amount);
    }

    /**
     * @dev Sets `amount` as the allowance of `spender` over the `owner` s tokens.
     *
     * This internal function is equivalent to `approve`, and can be used to
     * e.g. set automatic allowances for certain subsystems, etc.
     *
     * Emits an {Approval} event.
     *
     * Requirements:
     *
     * - `owner` cannot be the zero address.
     * - `spender` cannot be the zero address.
     */
    function _approve(
        address owner,
        address spender,
        uint256 amount
    ) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }

    /**
     * @dev Updates `owner` s allowance for `spender` based on spent `amount`.
     *
     * Does not update the allowance amount in case of infinite allowance.
     * Revert if not enough allowance is available.
     *
     * Might emit an {Approval} event.
     */
    function _spendAllowance(
        address owner,
        address spender,
        uint256 amount
    ) internal virtual {
        uint256 currentAllowance = allowance(owner, spender);
        if (currentAllowance != type(uint256).max) {
            require(currentAllowance >= amount, "ERC20: insufficient allowance");
            unchecked {
                _approve(owner, spender, currentAllowance - amount);
            }
        }
    }

    /**
     * @dev Hook that is called before any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
```

```
357         * will be transferred to `to`.
358         * - when `from` is zero, `amount` tokens will be minted for `to`.
359         * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
360         * - `from` and `to` are never both zero.
361         *
362         * To learn more about hooks, head to
            xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].
363         */
364        function _beforeTokenTransfer(
365            address from,
366            address to,
367            uint256 amount
368        ) internal virtual {}
369
370        /**
371         * @dev Hook that is called after any transfer of tokens. This includes
372         * minting and burning.
373         *
374         * Calling conditions:
375         *
376         * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
377         * has been transferred to `to`.
378         * - when `from` is zero, `amount` tokens have been minted for `to`.
379         * - when `to` is zero, `amount` of ``from``'s tokens have been burned.
380         * - `from` and `to` are never both zero.
381         *
382         * To learn more about hooks, head to
            xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].
383         */
384        function _afterTokenTransfer(
385            address from,
386            address to,
387            uint256 amount
388        ) internal virtual {}
389    }
390
```