// SPDX-License-Identifier: MIT pragma solidity ^0.6.2; import "./IERC20.sol"; import "./IERC20Metadata.sol"; import "./Context.sol"; import "./SafeMath.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.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226[How \* to implement supply mechanisms]. \* \* We have followed general OpenZeppelin guidelines: functions revert instead \* of 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 { using SafeMath for uint256; mapping(address => uint256) private \_balances; mapping(address => mapping(address => uint256)) private \_allowances; uint256 private \_totalSupply; string private \_name; string private \_symbol; /\*\* \* @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\_) public { \_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; } /\*\* \* @dev Returns the number of decimals used to get its user representation. \* For example, if `decimals` equals `2`, a balance of `505` tokens should \* be displayed to a user as `5,05` (`505 / 10 \*\* 2`). \* \* Tokens usually opt for a value of 18, imitating the relationship between \* Ether and Wei. This is the value {ERC20} uses, unless this function is \* overridden; \* \* NOTE: This information is only used for \_display\_ purposes: it in \* no way affects any of the arithmetic of the contract, including \* {IERC20-balanceOf} and {IERC20-transfer}. \*/ function decimals() public view virtual override returns (uint8) { return 18; } /\*\* \* @dev See {IERC20-totalSupply}. \*/ function totalSupply() public view virtual override returns (uint256) { return \_totalSupply; } /\*\* \* @dev See {IERC20-balanceOf}. \*/ function balanceOf(address account) public view virtual override returns (uint256) { return \_balances[account]; } /\*\* \* @dev See {IERC20-transfer}. \* \* Requirements: \* \* - `recipient` cannot be the zero address. \* - the caller must have a balance of at least `amount`. \*/ function transfer(address recipient, uint256 amount) public virtual override returns (bool) { \_transfer(\_msgSender(), recipient, amount); return true; } /\*\* \* @dev See {IERC20-allowance}. \*/ function allowance(address owner, address spender) public view virtual override returns (uint256) { return \_allowances[owner][spender]; } /\*\* \* @dev See {IERC20-approve}. \* \* Requirements: \* \* - `spender` cannot be the zero address. \*/ function approve(address spender, uint256 amount) public virtual override returns (bool) { \_approve(\_msgSender(), spender, amount); return true; } /\*\* \* @dev See {IERC20-transferFrom}. \* \* Emits an {Approval} event indicating the updated allowance. This is not \* required by the EIP. See the note at the beginning of {ERC20}. \* \* Requirements: \* \* - `sender` and `recipient` cannot be the zero address. \* - `sender` must have a balance of at least `amount`. \* - the caller must have allowance for ``sender``'s tokens of at least \* `amount`. \*/ function transferFrom( address sender, address recipient, uint256 amount ) public virtual override returns (bool) { \_transfer(sender, recipient, amount); \_approve(sender, \_msgSender(), \_allowances[sender][\_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance")); return true; } /\*\* \* @dev Atomically increases the allowance granted to `spender` by the caller. \* \* This is an alternative to {approve} that can be used as a mitigation for \* problems described in {IERC20-approve}. \* \* Emits an {Approval} event indicating the updated allowance. \* \* Requirements: \* \* - `spender` cannot be the zero address. \*/ function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) { \_approve(\_msgSender(), spender, \_allowances[\_msgSender()][spender].add(addedValue)); return true; } /\*\* \* @dev Atomically decreases the allowance granted to `spender` by the caller. \* \* This is an alternative to {approve} that can be used as a mitigation for \* problems described in {IERC20-approve}. \* \* Emits an {Approval} event indicating the updated allowance. \* \* Requirements: \* \* - `spender` cannot be the zero address. \* - `spender` must have allowance for the caller of at least \* `subtractedValue`. \*/ function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) { \_approve(\_msgSender(), spender, \_allowances[\_msgSender()][spender].sub(subtractedValue, "ERC20: decreased allowance below zero")); return true; } /\*\* \* @dev Moves tokens `amount` from `sender` to `recipient`. \* \* This is internal function is equivalent to {transfer}, and can be used to \* e.g. implement automatic token fees, slashing mechanisms, etc. \* \* Emits a {Transfer} event. \* \* Requirements: \* \* - `sender` cannot be the zero address. \* - `recipient` cannot be the zero address. \* - `sender` must have a balance of at least `amount`. \*/ function \_transfer( address sender, address recipient, uint256 amount ) internal virtual { require(sender != address(0), "ERC20: transfer from the zero address"); require(recipient != address(0), "ERC20: transfer to the zero address"); \_beforeTokenTransfer(sender, recipient, amount); \_balances[sender] = \_balances[sender].sub(amount, "ERC20: transfer amount exceeds balance"); \_balances[recipient] = \_balances[recipient].add(amount); emit Transfer(sender, recipient, amount); } /\*\* @dev Creates `amount` tokens and assigns them to `account`, increasing \* the total supply. \* \* Emits a {Transfer} event with `from` set to the zero address. \* \* Requirements: \* \* - `account` cannot be the zero address. \*/ function \_mint(address account, uint256 amount) internal virtual { require(account != address(0), "ERC20: mint to the zero address"); \_beforeTokenTransfer(address(0), account, amount); \_totalSupply = \_totalSupply.add(amount); \_balances[account] = \_balances[account].add(amount); emit Transfer(address(0), account, amount); } /\*\* \* @dev Destroys `amount` tokens from `account`, reducing the \* total supply. \* \* Emits a {Transfer} event with `to` set to the zero address. \* \* Requirements: \* \* - `account` cannot be the zero address. \* - `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); \_balances[account] = \_balances[account].sub(amount, "ERC20: burn amount exceeds balance"); \_totalSupply = \_totalSupply.sub(amount); emit Transfer(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 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 \* will be to transferred to `to`. \* - when `from` is zero, `amount` tokens will be minted for `to`. \* - when `to` is zero, `amount` of ``from``'s tokens will be burned. \* - `from` and `to` are never both zero. \* \* To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks]. \*/ function \_beforeTokenTransfer( address from, address to, uint256 amount ) internal virtual {} }