```solidity
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.8.0) (token/ERC721/IERC721.sol)

pragma solidity ^0.8.0;

import "../../utils/introspection/IERC165.sol";

/**
 * @dev Required interface of an ERC721 compliant contract.
 */
interface IERC721 is IERC165 {
    /**
     * @dev Emitted when `tokenId` token is transferred from `from` to `to`.
     */
    event Transfer(address indexed from, address indexed to, uint256 indexed tokenId);

    /**
     * @dev Emitted when `owner` enables `approved` to manage the `tokenId` token.
     */
    event Approval(address indexed owner, address indexed approved, uint256 indexed tokenId);

    /**
     * @dev Emitted when `owner` enables or disables (`approved`) `operator` to manage all of its assets.
     */
    event ApprovalForAll(address indexed owner, address indexed operator, bool approved);

    /**
     * @dev Returns the number of tokens in ``owner``'s account.
     */
    function balanceOf(address owner) external view returns (uint256 balance);

    /**
     * @dev Returns the owner of the `tokenId` token.
     *
     * Requirements:
     *
     * - `tokenId` must exist.
     */
    function ownerOf(uint256 tokenId) external view returns (address owner);

    /**
     * @dev Safely transfers `tokenId` token from `from` to `to`.
     *
     * Requirements:
     *
     * - `from` cannot be the zero address.
     * - `to` cannot be the zero address.
     * - `tokenId` token must exist and be owned by `from`.
     * - If the caller is not `from`, it must be approved to move this token by either {approve} or {setApprovalForAll}.
     * - If `to` refers to a smart contract, it must implement {IERC721Receiver-onERC721Received}, which is called upon a safe transfer.
     *
     * Emits a {Transfer} event.
     */
    function safeTransferFrom(
        address from,
        address to,
        uint256 tokenId,
        bytes calldata data
    ) external;

    /**
     * @dev Safely transfers `tokenId` token from `from` to `to`, checking first that contract recipients
     * are aware of the ERC721 protocol to prevent tokens from being forever locked.
     *
     * Requirements:
     *
     * - `from` cannot be the zero address.
```

```
68          * - `to` cannot be the zero address.
69          * - `tokenId` token must exist and be owned by `from`.
70          * - If the caller is not `from`, it must have been allowed to move this token by
            either {approve} or {setApprovalForAll}.
71          * - If `to` refers to a smart contract, it must implement
            {IERC721Receiver-onERC721Received}, which is called upon a safe transfer.
72          *
73          * Emits a {Transfer} event.
74          */
75         function safeTransferFrom(
76             address from,
77             address to,
78             uint256 tokenId
79         ) external;
80
81         /**
82          * @dev Transfers `tokenId` token from `from` to `to`.
83          *
84          * WARNING: Note that the caller is responsible to confirm that the recipient is
            capable of receiving ERC721
85          * or else they may be permanently lost. Usage of {safeTransferFrom} prevents
            loss, though the caller must
86          * understand this adds an external call which potentially creates a reentrancy
            vulnerability.
87          *
88          * Requirements:
89          *
90          * - `from` cannot be the zero address.
91          * - `to` cannot be the zero address.
92          * - `tokenId` token must be owned by `from`.
93          * - If the caller is not `from`, it must be approved to move this token by
            either {approve} or {setApprovalForAll}.
94          *
95          * Emits a {Transfer} event.
96          */
97         function transferFrom(
98             address from,
99             address to,
100            uint256 tokenId
101        ) external;
102
103        /**
104         * @dev Gives permission to `to` to transfer `tokenId` token to another account.
105         * The approval is cleared when the token is transferred.
106         *
107         * Only a single account can be approved at a time, so approving the zero address
            clears previous approvals.
108         *
109         * Requirements:
110         *
111         * - The caller must own the token or be an approved operator.
112         * - `tokenId` must exist.
113         *
114         * Emits an {Approval} event.
115         */
116        function approve(address to, uint256 tokenId) external;
117
118        /**
119         * @dev Approve or remove `operator` as an operator for the caller.
120         * Operators can call {transferFrom} or {safeTransferFrom} for any token owned by
            the caller.
121         *
122         * Requirements:
123         *
124         * - The `operator` cannot be the caller.
125         *
126         * Emits an {ApprovalForAll} event.
127         */
128        function setApprovalForAll(address operator, bool _approved) external;
129
130        /**
131         * @dev Returns the account approved for `tokenId` token.
132         *
```

```
133         * Requirements:
134         *
135         * - `tokenId` must exist.
136         */
137        function getApproved(uint256 tokenId) external view returns (address operator);
138
139        /**
140         * @dev Returns if the `operator` is allowed to manage all of the assets of
            `owner`.
141         *
142         * See {setApprovalForAll}
143         */
144        function isApprovedForAll(address owner, address operator) external view returns (
           bool);
145    }
146
```