# CRC Cards

All instance variables are private unless stated otherwise. All instance methods are public unless stated otherwise.

- Gettable → (has a getter function)
- Settable ← (has a setter function)
- Both ⇄

To place a function, it **must** end with (). Because we use → and ← to denote the existence of getters and setters, they will be ignored.

# Interfaces

| **ReprAble**<br>*This interface represents all Classes that can export a string representation of itself in a way that is unambiguous, such that it can be directly loaded back into the class or another instance of the class.*<br>Superclasses:<br>Subclasses:<br>Implements:<br>Type: Interface | |
|---|---|
| Responsibilities | Interacts with the following classes |
| • repr()<br>  ○ Returns this Class in a string representation such that it can be read back.<br>• setFromRepr()<br>  ○ The inverse of repr(). Classes implementing this must be able to reconstruct any instance that may ever exist simply with this method. | • BanStatus<br>• DateEntries<br>• Permissions |

# Entities

| **Account**<br>*An account.*<br>Superclasses:<br>Subclasses:<br>Implements:<br>Type: Entity | |
|---|---|
| Responsibilities | Interacts with the following classes |

| | |
|---|---|
| ● username → ● password ⇄ ● account history → ● permissions → ● banStatus → ● addNowToAccountHistory() ○ Adds the current time to the account history | Dependencies: ● DateEntries (account history) ● BanStatus ● Permissions These classes are dependent on this class: ● AccountCreator ● AccountLogin ● AccountManager ● StorageLoader ● StorageManager |

| **BanStatus** *Tracks whether the user is banned or not and manages banned expiration dates.* Superclasses: Subclasses: Implements: ReprAble Type: Entity | |
|---|---|
| Responsibilities | Interacts with the following classes |
| ● bannedUntil ● ban(until) ○ Bans user until specified date ● isBanned() ○ Returns a boolean value determining whether someone is still banned ● unBan() ○ Unbans the user | Dependencies: ● None These classes are dependent on this class: ● Account ● All use cases that are dependent on Account |

| **Permissions** *Tracks whether the user is banned or not and manages banned expiration dates.* Superclasses: Subclasses: Implements: ReprAble Type: Entity | |
|---|---|
| Responsibilities | Interacts with the following classes |
| ● permissions ○ Store all the user permissions ● toString() ● getPermissions() | Dependencies: ● None These classes are dependent on this class: ● Account ● StorageLoader |

| | |
|---|---|
| ● hasPerm() ○ Has a counterpart that can check multiple permissions, if needed ● addPerm() ● removePerm() ● clearPerms() | ● AccountManager |

| | |
|---|---|
| **DateEntries** *Account login history tracker* Superclasses: Subclasses: Implements: ReprAble Type: Entity | |
| Responsibilities | Interacts with the following classes |
| ● dates ● addDate(curDate) ● clearDates() | Dependencies: ● None These classes are dependent on this class: ● Account ● StorageLoader ● StorageManager |

| | |
|---|---|
| **Today (static methods only)** *This class has a static method that returns the current time.* Superclasses: Subclasses: Implements: Type: Entity | |
| Responsibilities | Interacts with the following classes |
| ● getToday() | ● Account ● BanStatus |

| |
|---|
| **AccountStorage** *Tracks existing accounts* Superclasses: Subclasses: Implements: Type: Entity |

| Responsibilities | Interacts with the following classes |
|---|---|
| ● accounts<br>● addAccount(account)<br>● removeAccount(username \| account)<br>● checkAccontExists(username \| account)<br>● getAccount(username) | Dependencies:<br>● Account<br>These classes are dependent on this class:<br>● StorageManager |

## Use cases

| **AccountCreator**<br>*Creates new accounts*<br>Superclasses:<br>Subclasses:<br>Implements:<br>Type: Use case | |
|---|---|
| Responsibilities | Interacts with the following classes |
| ● accountStorageManager<br>● createAccount(username \| password) | Dependencies:<br>● Account<br>● StorageManager<br>These classes are dependent on this class:<br>● AdminAccountManager<br>● Controller |

| **AccountLogin**<br>*Allows logging into accounts*<br>Superclasses:<br>Subclasses:<br>Implements:<br>Type: Use case | |
|---|---|
| Responsibilities | Interacts with the following classes |
| ● accountStorageManager<br>● login(username \| password)<br>● validateLogin(username \| password) | Dependencies:<br>● Account<br>● StorageManager<br>These classes are dependent on this class:<br>● Controller |

| **AccountManager** |
|---|

| *Manages standard user accounts*<br>Superclasses:<br>Subclasses:<br>Implements:<br>Type: Use case | |
|---|---|
| Responsibilities | Interacts with the following classes |
| ● account<br>● commandList<br>● accountStorageManager<br>● validatePermission(permission)<br>● validatePermissions(permissions)<br>● setPassword(old, new) | Dependencies:<br>● Account<br>● Permissions<br>● StorageManager<br>These classes are dependent on this class:<br>● Controller |

| **AdminAccountManager**<br>*Manages admin user accounts*<br>Superclasses: AccountManager<br>Subclasses:<br>Implements:<br>Type: Use case | |
|---|---|
| Responsibilities | Interacts with the following classes |
| ● createNewAdminUser(username, password)<br>● addPermission(username, password)<br>● revokePermission(username, password)<br>● banUser(username, unbanDate)<br>● unbanUser(username)<br>● deleteUser(username) | Dependencies:<br>● Account<br>● Permissions<br>● Date<br>These classes are dependent on this class:<br>● ControllerInputAdmin<br>● ControllerInputStandard |

| **StorageLoader**<br>*Loads and exports accounts from and to a .csv file*<br>Superclasses:<br>Subclasses:<br>Implements:<br>Type: Use case | |
|---|---|
| Responsibilities | Interacts with the following classes |
| ● accountStorage (protected)<br>● username<br>● password | Dependencies:<br>● AccountStorage<br>● Account |

| | |
|---|---|
| ● permissions<br>● banStatus<br>● accountHistory<br>● loadAccount()<br>  ○ Adds and account object to the system from a string value<br>● createAccount()<br>  ○ Creates an account in the csv file based on the information given<br>● updateAccount()<br>  ○ Updates the account information stored in the csv file<br>● copyFile(File, File)<br>  ○ Makes a copy of the file<br>● updateAccounts(accountStorage)<br>  ○ Exports the account storage passed in to a CSV file. | ● Permissions<br>● BanStatus<br>● DateEntries (account history log)<br>These classes are dependent on this one:<br>● StorageManager |

| **StorageManager**<br>*Tracks existing accounts*<br>Superclasses:<br>Subclasses:<br>Implements:<br>Type: Use case | |
|---|---|
| Responsibilities | Interacts with the following classes |
| ● The constructor checks if there’s an existing CSV file to load data from. If so, load it. Otherwise, start with an empty account storage.<br><br>● accountStorage<br>● addAccount(account)<br>● removeAccount(username \| account)<br>● checkAccountExists(username \| account)<br>● getAccount(username)<br>● updateSave()<br>  ○ Export everything in accountStorage to a CSV file. | Dependencies:<br>● AccountStorage<br>● Account<br>These classes are dependent on this class:<br>● Controller<br>● AccountCreator<br>● AccountLogin<br>● AccountManager |

## Controllers

**Definition (Window).** A window has a set of commands a user may execute. Different windows may print different prompts to the screen, and may have different commands.

**ControllerInput**
*An abstract class with methods and variables for interacting with a LOGGED IN user*
Superclasses:
Subclasses: ControllerInputAdmin, ControllerInputStandard
Implements:
Type: Controller

| Responsibilities | Interacts with the following classes |
|---|---|
| ● curState →<br>    ○ This shows the “window” the program should display next **after** the current controller has finished executing.<br>● ControllerInput(accountManager, accountStorageManager, presenter)<br>● inputParser(input) | Controller<br>ControllerInputStandard |

**ControllerInputAdmin**
*Admins should have access to this controller.*
Superclasses: ControllerInputStandard
Subclasses:
Implements:
Type: Controller

| Responsibilities | Interacts with the following classes |
|---|---|
| ● inputParser(input) | Presenter<br>AccountManager<br>AdminAccountManager<br>StorageManager |

**ControllerInputStandard**
*Standard users should have access to this controller.*
Superclasses:
Subclasses: ControllerInputAdmin
Implements: ControllerInput
Type: Controller

| Responsibilities | Interacts with the following classes |
|---|---|
| ● inputParser(input) | Presenter<br>AccountManager<br>StorageManager |

| **Controller**<br>*Interacts with the user and calls necessary use case methods*<br>Superclasses:<br>Subclasses:<br>Implements:<br>Type: Controller | |
|---|---|
| Responsibilities | Interacts with the following classes |
| <ul><li>presenter (package-protected)<ul><li>This stores the presenter class.</li></ul></li><li>accountStorageManager<ul><li>This stores the account storage manager, the system used to import and export accounts.</li></ul></li><li>manager<ul><li>This stores the account manager as a class.</li></ul></li><li>loginState →<ul><li>Lets us know whether the controller thinks a user is logged in or not.</li></ul></li><li>loggedInState →<ul><li>Which "window" to show to the user.</li></ul></li></ul>These two methods below are private.<ul><li>register(username, password)</li><li>login(username, password)</li></ul>**LOOPS**<br>These methods are repeatedly called throughout the program.<ul><li>**loggedOutMenu(input)**<ul><li>Input loop for logged out users</li><li>A user is prompted to type something before this method is run. What the user types will be passed into the *input* of this method. If the user did not log in after this method is called, the program will re-prompt the user to type something and this method is called again.</li><li>**A user will always be prompted THIS when running the program at first.**</li></ul></li><li>**inputParser(input)**</li></ul> | Presenter<br>ControllerInput & all its inheritors<br>AccountCreator<br>AccountLogin<br>AccountManager<br>StorageManager |

| | |
|---|---|
| ○ Input loop for logged in users. A user is asked to input something, likely from a choice of words, and this method reacts to such input. **This method is the method that interacts with all ControllerInputs.** The inputs this method accepts will depend on loggedInState. Once this method completes, the user is asked to input something again and this method is called again, and so on.<br>○ There is no way to log a user out once the program is in this loop without stopping the program. | |

## Presenters

| **Presenter**<br>*Presents information onto the command line and determines the text that gets presented*<br>Superclasses:<br>Subclasses:<br>Implements:<br>Type: Presenter | |
|---|---|
| Responsibilities | Interacts with the following classes |
| ● startView()<br>○ Prints a message to a screen for users who are logged out.<br><br>● printAndAskPrompt()<br>○ For logged in users, based on which “window / state” a logged-in user is in, prints out something and asks for user input. This method returns what the user wrote.<br>● dashboardView()<br>○ Prints the message passed into the method and requests for user input, which it returns.<br><br>These methods ask the user to type in something and returns a string or an array of strings based on what the user typed. | Controller |

- enterCredentials()
    - Prompts the username and password and returns both when finished.
- enterUsername()
    - Prompts the username only for any reason.
- enterDate()
    - Prompts the user to enter **a date.** Throws an exception if what the user entered isn't a date.

- accountHistoryView()
    - Displays account history when asked to.

All these methods affirm that an action has been done to a user. It prints to the user whether the following action was successfully enacted to the user.

- banUserConfirm()
- deleteUserConfirm()
- createNewAdminConfirm()
- promoteUserConfirm()