**\*\*\*Anything highlighted in orange will NOT be turned in as part of phase\_1, it was the first go around. Sections 3 an onwards WILL be turned in \*\*\***

1. General Architecture (\*): this will contain all "classes", member variables, and methods of our architecture.

2. Constraints: this will contain all constraints in pseudocode taken from the document.

3. Data types (inserting data types into the IFD)

4. Business logic constraints

5. Abstract Code

\* please note that a \* to the right of a property indicates it is a user-defined class, not a primitive type. For example, User.location is an instance of Location which we will also create.

\* please note that member variables will be in the syntax x.y and member methods will be in the syntax x.y(). We are not worrying about parameters at this phase, just the general abstraction.

1. GENERAL ARCHITECTURE (classes/entities/methods/member variables):

-----------------------

The database:

System             // this will be used architecturally as the Database in this AC/pseudocode.

System.Users            // this db table will correspond with the User class

System.Transactions

System.Items

System.Searches

System.GameTypes        // should be updatable in system/db, but not from GUI per document

The code:

User                // general user

User.email

User.password

User.firstName

User.lastName

User.nickName

User.location\*            // used at registration time, not to be confused with searching location

User.phoneNumber

User.phoneNumberType    // home, work or mobile

User.showPhoneNumber    // show phone number on swaps, an option while registering phone#

User.rating

User.listedItems[]\*        // array or list of type Item

User.allItems[]\*

User.addItem(Item item)

User.removeItem(Item item)

User.updateInfo()        // updating user information

User.isAvailableForSwap()

User.isEmailAvailable()    // since an email can only be registered once

User.isPhoneNumberAvailable()    // phone number can only be registered to one user

Session            // class for managing a persons session while browsing the GUI

Session.loggedIn        // boolean value true if a person is logged in successfully

Session.user\*

Session.currentLocation\*    // can be used for searching, this field will need to be filled in by user

Session.login()

Session.register()

Session.logout()

Session.mainMenu()

Session.viewItem()

Session.listItems()

Session.myItems()

Session.searchItems()

Session.swaps()        // page to accept/reject swaps

Session.swapDeatils()

Session.swapHistory()

Session.updateInfo()

Session.postMessage()    // this will be used to signal a warning, or any other gui 'message'

Session.proposeSwap()

Sesssion.confirmSwap()

Location        // a class to organize components of a location in one place

Location.city

Location.state

Location.postalCode

Location.latitude

Location.longitude

Item

Item.name

Item.listingNumber        // System will track/set this, it is ordinal basaed on when item was input

Item.gameType        // board game, card game, video game, computer game, or jigsaw puzzle

Item.details        // HashMap<String,String> based on type of game, can have more details

Item.condition

Item.description    // optional

Item.hasPreviousSwap()

Item.hasSwapPending()

Item.myHistory\*        // array/or list with dates and of type Swap, which then of course has users

Search

Search.user\*        // the User conducting the search

Search.searchType    // keyword, within user's postal code, within x miles of user, within specific postal code

Search.startingPostal

Search.searchNumberMilesFromMe()

Search.searchKeyword()

Swap

Swap.proposalDate        // type Date. needed for swap history

Swap.finalStatusDate        // type Date. needed for swap history

Swap.proposer\*        // type User

Swap.counterparty\*        // type User

Swap.proposerItem\*        // type Item

Swap.counterpartyItem\*    // type Item

Swap.finalStatus        // completed or rejected. once completed, we can enable 'onCompleted()'

Swap.onCompleted()        // when swap is successful, we need to: see contact info for proposer.

Swap.ratingEnabled()        // if true, users should be able to rate each other from swap history and ..

2. CONSTRAINTS (included also, GUI notes (i.e. make text red when rating < x%))

--------------

user.setPhoneNumber.setToOptional();

phoneNumber.availableTypes = [{"home", "work", "mobile"}]

given String email:

if(System.contains(email)) return ERROR;

given String phoneNumber and String userName:

if(System.contains(phoneNumber)) return ERROR; //TODO: do we need to check anything more specifically here?

item.availableGameTypes = [{"BoardGame","CardGame","VideoGame","ComputerGame","JigsawPuzzle"}]

item.availableConditionTypes = [{"Mint","LikeNew","LightlyUsed","ModeratelyUsed","HeavilyUsed","Damaged/MissingParts"

item.setDescription is optional

given int itemListingNumber:

if(System.contains(itemListingNumber)) return ERROR;

given Item item and an attempted Swap swap:

if(item.isPending()) return ERROR;

if(item.hasCompletedSwap()) return ERROR;

if(swap.counterparty==swap.proposer) return ERROR;    // can't swap with self

if(swap.accepted()) swap.showProposerContactInfl();

if(swap.rejected()) swap.disableFutureProposalOfThisType();

if(swap.completed()) swap.enableRating();

// Logging In / Registering :

if(login.hasValue && phoneNumber.hasValue) return ERROR 'Please use either email or phone num';

if(password.isEmpty()) return ERROR 'Please enter pw';

if(loginButton.clicked()) {

    boolean errorExists = false;

    String message = "";

    if(emailNotRegistered) message += emailNotRegisteredError; errorExists=true;

    if(phoneNumNotRegistered) message += phoneNumNotRegisteredError; errorExists=true;

    if(password.isIncorrect()) message += incorrectPwError; errorExists=true;

    if(errorExists) {

        Session.showError(message);

        return ERROR;

    } else { Session.login(); //successful login

        Session.showWelcome();

    }

}

//TODO: add a postal code verification

// onced logged in:

if(me.unacceptedSwaps>0) {

    if(me.unacceptedSwaps>5 || me.hasSwapOlderThanFiveDays()) {

        showAcceptReject(warning); }

    else showAcceptReject();

}

if(me.unratedSwaps>0) {

    if(me.unratedSwaps>2) {

        showRateSwap(warning); }

    else showRateSwap();

}

if(me.isEmpty(rating)) gui.showRating("None");

else me.rating.roundToHundredths();

// listing an item:

if(me.unratedSwaps>2 || me.unacceptedSwaps>5) {

    Session.disableNewListing(); //this willl also show the message;

} else continue...;

switch(item.gameType):

    case "VideoGame":

        showDropdown("Title: Platform","Nintendo","Playstation","Xbox");

        showDropdown("Title: Media","optical disc", "game card", "cartridge");

    case "ComputerGame":

        showDropdown("Title: Platform","Linux","macOS","Windows");

    case "JigsawPuzzle":

        showDropdown("Piece count");

    default:

        break;

if(newListing.saveButtoClicked()) {

    doErrorChecking;

    listingNum = System.getNextListingNumber();

    Item item = new Item(listingNum);

    Session.currentUser.addItem(thisItem); // this will call User.addItem();

    Session.showSuccessListingMessage('Your item has been listed! Your num is'+listingNum);

}

// My Items:

if(descrition.length>100) {

    gui.setDescription(description.substr(0,100)+"...");

}

// Searching:

if(postalCode.isInvalid()) return ERROR;    //i.e. 5 integers

if(!withinXMiles.isInteger()) return ERROR;

if(searchResults.isEmpty()) showMessage("Sorry, no results found!");

// see same constraint with description in "my items" section, applies here too.

distanceFromUser.roundToTenths();

if(searchType=="Keyword") {

    for(String s : Item.description) {

        if(search.keyword==s) {

            Gui.description.findWord(s).setBackground("blue");

        }

    }

}

// Viewing items:

if(item.description.isEmpty()) GUI.hideFieldAltogether();

if(item.getUser()!=me) //display nickname, city, state, postal code, swapper rating (rounded to 100, and "none if nonexistent), distance from user rounded to tenths (unless same postal code)

if(item.getUser()!=me && item.getZip()!=me.getZip()){

    if(item.distanceFromMe() >0.0 && item.distanceFromMe() <= 25.0) {

        GUI.highlightDistance("green");

    else if(item.distanceFromMe() >25.0 && item.distanceFromMe() <=50.0) {

        GUI.highlightDistance("yellow");

    else if(item.distanceFromMe() >50.0 && item.distanceFromMe() <= 100.0) {

        GUI.highlightDistance("orange");

    else GUI.highlightDistance("red");

}

if(item.getUser().unratedSwaps()<3 && item.getUser().unacceptedSwaps < 6 && item.available()) {

    GUI.showProposeSwapOption();

}

// Proposing a swap:

if(me.cannotProposeSwaps()) return ERROR; //in reality we shouldn't even be able to get here

if(item.getUsers().distanceFromMe()>=100.0) GUI.show100MileWarning();

if(item.successfulSwapProposed()) GUI.showConfirmation(); GUI.allowReturMainMenu();

if(swap.isAccepted()) {

    GUI.showProposerInfo();    // this method should check if there is phone number. if not, show "No phone number available"

    GUI.removeSwapFromListing();

}

if(swap.isRejected()) {

    swap.finalStatus="rejected";    //means this swap CAN'T be proposed again.

}

if(!swap.finalStatus.isEmpty()) // meaning when a swap is completed {

    GUI.rateSwaps();

    swap.allowRating(); //0 -5 ordered by acceptance date

}

if(swap.isRated()) {

    GUI.rateSwaps().remove(swap);    // once rating complete, remove it from this GUI portion

}

// Swap History :

user.pctRejected.round(.0%);

if(user.pctRejected > 50.0%) {

    GUI.highlightPctRejected("red");

}

if(!swap.isRated() && !swap.finalStatus.isEmpty()) // a completed swap that hasn't been rated yet {

GUI.showOptionForRating();

}

// Update user info:

GUI.blankOutUpdatingEmail();

if(user.unapprovedSwaps>0 || user.unratedSwaps>0) {

    Session.disableUpdatingInfo();

}

if(user.changePhoneNumber()) {

    System.scanForPhoneNubmer();

    if(found) throw ERROR; GUI.showError();

}

3. DATA TYPES

--------------

**User**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Data type** | **Nullable** |
| email | String | Not Nullable |
| password | String | Not Nullable |
| firstName | String | Not Nullable |
| lastName | String | Not Nullable |
| nickName | String | Nullable |
| location | Location \*(custom type) | Not Nullable |
| phoneNumber | String | Nullable |
| phoneNumberType | String | Nullable |
| showPhoneNumber | Boolean | Not Nullable |
| rating | Double | Not Nullable |
| listedItems | List<Item> \*(custom type) | Not Nullable |
| allItems | List<Item> \*(custom type) | Not Nullable |
| isAvailableForSwap | Boolean | Not Nullable |

**Location**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Data type** | **Nullable** |
| city | String | Not Nullable |
| state | String | Not Nullable |
| postalCode | String | Not Nullable |
| latitude | Double | Nullable |
| longitude | Double | Nullable |

**Item**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Data type** | **Nullable** |
| name | String | Not Nullable |
| listingNumber | Long | Not Nullable |
| gameType | String | Not Nullable |
| details | String | Not Nullable |
| condition | String | Not Nullable |
| description | String | Nullable |
| hasPreviousSwap | Boolean | Not Nullable |
| hasSwapPending | Boolean | Not Nullable |
| mySwapHistory | List<Swap> \*(custom type) | Not Nullable |

**Swap**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Data type** | **Nullable** |
| proposer | User \*(custom type) | Not Nullable |
| counterparty | User \*(custom type) | Not Nullable |
| proposalDate | Date | Not Nullable |
| finalStatusDate | Date | Not Nullable |
| proposedItem | Item \*(custom type) | Not Nullable |
| counterpartyItem | Item \*(custom type) | Not Nullable |
| finalStatus | String | Not Nullable |
| ratingEnabled | Boolean | Not Nullable |

4. BUSINESS LOGIC CONSTRAINTS

--------------

Underlined items are forms or GUI aspects.

*Italicized items are “classes” or attributes of classes.*

**Bold are super-sections.**

**User**

* When accessing the app initially, all *users* will be directed to the login form.
* Users can login with an *email* or *phone number*, and *password*.
* A register link must be provided on the login form.
* *Phone type* is limited to either home, work, or mobile.
* An *email* can only be registered once in the system. Check if the email the user is attempting to register already exists in the database.
* If *number of proposed swaps* user has received is greater than zero, display link to accept/reject swap form.
  + If this is greater than five, the number should be printed in bold and in red.
* If *number of unrated swaps* user has is greater than zero, display link to swap rating page.
  + If this is greater than two, the number should be printed in bold and red and user can not propose another swap.
* *Rating* must be rounded to hundreths, or display “None” if there is none.
* *Rating,* when entered after a completed swap, is a choice from 0 to 5 (whole numbers).
* Logout  button must be visible which returns the *user* to the login form.
* If a user has any unapproved swaps or unrated swaps, can not make any updates to user info.
* If a user is trying to update user info, they can not update their email address only.
* If a user attempts to change phone number but it is already in use, appropriate error message shown.

**Item**

* *Game type* must be one of: board game, card game, video game, computer game, jigsaw puzzle.
* If *game type* is jigsaw puzzle:
  + Attribute *piece count* is now present.
* If *game type* is computer game:
  + *Platform* is limited to choices of Linux, macOS, or Windows.
* If *game type* is video game:
  + *Platform* must be listed as Nintendo, Playstation, or Xbox. This list of values should be updatable in DB (not necessary to update it in the GUI).
* *Game condition* must be one of: mint, like new, lightly used, moderately used, heavily used, or damaged/missing parts.
* User-entered *description* is optional.
* If a user has more than two unrated swaps, or more than five unaccepted swaps, they can not list a new item.
* Only fields needed for chosen game are displayed.
* When viewing item, if no description exists, don’t show the field.
* If viewing item belonging to another user:
  + Their nickname, city, state, postal code should be displayed with swapper rating rounded to hundreths and distance to user rounded in tenths (unless in same postal code in which case the field shouldn’t be displayed).
  + If distance is between 0 and 25.0 miles, highlight with a green background.
  + If distance is between 25.0 and 50.0 miles, highlight with a yellow background.
  + If distance is between 50.0 and 100.00 miles, highlight with an orange background.
  + If distance is over 100.0 miles, highlight with a red background and warning message shown.

**Search**

* Searches should be available by keyword, postal code, within x miles, and within a specific postal code.
* If description is greater than 100 characters, show ellipsis at end.
* If keyword search, match keyword highlighted with blue background in the results.
* Detail button should be provided to show item’s specifics.
* If within a specific postal code:
  + Ensure postal code is valid.
* If within x miles:
  + X must be a user-entered whole number.

**Swap**

* Items that are associated with a pending swap are not available to swap.
* Items that are associated with completed swaps are not available to swap.
* If a user has a listed item, they are able to swap items with each other.
* If a swap is rejected, that specific item-for-item swap can not be proposed again.
* If a swap is accepted, the counterparty will see the contact info for the proposer.
* To mark swap as completed, after swapping items, users must rate each other on a scale of 0-5.
* Items that have been swapped cannot be swapped again.
* If % of swaps rejected historically is greater than 50.0%, highlight number in red.
* If users have not rated each other yet, show Rating option in swap history page.

5. ABSTRACT CODE

--------------

Underlined items are forms, buttons, or other GUI items.

*Italicized items are “classes” or attributes of classes.*

**Bold are super-sections.**

**Login**

* *User* enters an *email* or *phoneNumber*, and a *password*.
* Data validation occurs for all inputted fields.
* User hits Register button:
  + Go to Registry
* *User* hits Enter button:

If (*email* != empty AND *phoneNumber*!=empty):

Error(“Login with either email or phone number”)  
If(*email*==found AND !*password*.verify())

Error(‘Incorrect password for ‘+email)

If(*phoneNumber*==found AND !*password*.verify())

    Error(‘Incorrect password for ‘+phoneNumber)

    if((*email*==found AND *password*.verify()) OR (*phoneNumber*==found AND *password*.verify())

        Go to Main Menu

**Registry**

* Display text boxes for the following fields: *email, nickname* (optional), *city, first name, last name, state, postal code, phone number* (optional).
* Display password box (any inputted text is masked by ‘\*’) for field *password*.
* Display *Register* button at bottom.
* *User* hits *Register button*:
  + Do data validation.
  + Check for

**Main Menu**