Amr, Javier, and Elena

- **High-Level Description:**
  - Laundry app for students at Amherst
  - Features
    - Constantly updated schedule of what washers/dryers are available and the time remaining, as well as the user that has a load in the machine. They can also be pending, meaning they are done but the load has not been picked up by the owner yet.
    - Put a load in: this feature is so that users can mark on the app when they have occupied a machine to let their peers know. The user has one load, which can be put in a machine if it is available and if it's not already in another machine. Then, once it is washed, you can mark it as put in the dryer. The users should also let people know when they take the load out of the machine. Additionally, if a person takes more than 10 minutes to take their load out, then anyone can take the load out for them.
    - Bubble tracker (social score)
      - Increase bubbles when laundry is taken out on time
      - Decrease after you're late for the loads
      - If you complete a load in any machine, you will get 100 bubbles. If you pick it up more than 10 minutes late, you will lose 15 bubbles. An hour late → lose 100 bubbles. 3 hours late → lose 200 bubbles. A day late → lose 500 bubbles.
      - We have a leaderboard of the people with the most bubbles to promote good laundry practice.
    - Queue: This works so that if you really want to wash and all the machines are taken, then you can join the queue and when any machine opens, the next person in the queue will have 10 minutes reserved for them to put their load in the machine, else, the next person in the queue might take their spot. We have a queue for both the dryers and the washers.
  - The program uses data based on a json file, which includes both fake users that we made and users we have created while testing the app. This file includes information about the user's profile, the status of every machine and whose load it has inside, and also the people that are currently in the queue. We used the json-simple-1.1.jar library to make the methods that read and write the data.
  - Additionally, we learned how to use networks so that if we are running a server, it can hold a thread for every user that connects to it via a socket. Then, the input and output data streams are set to be the json textfile with the data, so that every time any user makes any change, they write the data, send it over to the server, and the server sends it back to every user. Every user is also constantly updating their data.

- Before running, the user has to register by creating a new user that would be added into the data or by putting in the email of an already existing user, which would log you in as that user with the progress stored in data.

❖ **Specifics:**

Classes:

- Project extends jpanel implements mouse listener, action listener:
    - Fields
        - World instance ( where all the data is stored and everything happens)
        - Height, width, fps (to set up jpanel)
        - Int page - stores information on which page the user is currently looking at in the app.
        - JSONObject data - an object that our library makes from the text file to help parse the data.
        - User you - points at the user that is currently logged in.
        - BufferedImages of all the pictures that we use for out GUI.
        - JTextField - 3 of them which is where the user inputs email, name, last name when logging in or signing up.
        - Strings - 3 of them that signify the input of what the user puts in the text fields.
        - Load deadLoad - it is the load that gets assigned to a machine when it is emptied. Before, we used to set the load to null, but this ended up setting the user's version of the same load as null as well, so instead we made the machine point to this load.
        - DataOutput/Input Stream - points at the socket where it needs to send or receive a file from.
        - Boolean notFound helps tell the program if the account input by the user exists.
    - Methods
        - public static void main(String args[ ])
            - Sets up Jframe and all the network semantics.
            - Creates project instance.
        - Project constructor(Jframe)
            - Makes the world instance
            - Adds mouselistener
            - Finishes setting up jFrame, and creates the runner thread that constantly repaints.
            - Makes and positions all of the registration text boxes and sends

the user to that page.

● Tries to initialize all of the buffered images from files.

■ sendFile(String path)

● Makes a new file with the json file address and writes it in a byte array to send it over through the outputstream to the server. ■ receive File(path)

● Similarly, reads the incoming file as a byte array and then writes it on a fileOutputStream object.

● These 2 methods are both static and void as don't return anything because it just puts the new file in the project directory but doesn't change anything in the program itself.

■ paintcomponent(graphics g)

● Void as it only produces side effects and doesn't return anything. ● Goes through a switch clause that depends on the page field, so it knows which page to draw.

● From our design, because what happens is depending on where the user clicks, this method and the mouse listening method are one of the most important ones - where everything happens and it takes use of all other methods from other classes to know what to draw or change within the app and data.

● Chooses colors, fonts from their respective classes.

● For page 1, home page, it prints the users name, bubbles, all the buttons to the other pages and washer and dryer availability ● Page 2 is the machine status, uses a bunch of if statements based on all the machines to determine if it needs to show a green light if available, a red light if occupied, (if so, it calls time remaining method to print that out and the owner of the load) or yellow if it is pending (hasn't been taken out).

● Page 3: similarly, also goes through many if statements that based on semantics decides whether to draw ‘register load’, ‘unavailable’ or ‘take load out’ buttons. It goes through each machine and considers aspects like if the machines load is the deadLoad, if you are the owner of the load in it, if the time remaining is less than 0. It also looks at if the user’s load is already washed, if so, it prints a go to dryers page button.

● Page 4: it is the dryers page. Works in the exact same way as the previous page but it is for the dryers instead of washers.

● Page 5: just draws the image for it as nothing happens here, it is an informational washing tips page.

- Page 7: Prints the queue page, which only has two buttons that you can click to take you to either the washing queue or the drying queue.

- Page 8: it is the leaderboard, prints out the top 10 people with bubbles based on the World leaderboard field. The top person has a bigger font than the other people. It uses a loop to print everyone down in a vertical fashion. Uses a multiplier and considers the Strings' length to prints them in the center
- Page 9: is the profile page, prints the information and stats for the current user, uses the getter methods in user to find what to print. Also checks the booleans the user's load to print the machine status of the user.
- Page 10: prints the dryer queue in a vertical fashion and the button to join it.
- Page 11: same but with the washer queue.
- Page 12: the new user registration page, makes the text fields visible
- Page 13: It is the returning user page. Only makes the email text box visible. Checks the boolean not found to see whether it prints account not found.
- Page 14: the initial page, where you choose if you are a returning user or a new user.

■ Action performed(Action Event e)

- Takes the text from the input from the text fields.
- Reads the data so that the info the user provides is not overwritten later when reading.

● Considers if you re in page 12 or 13 to know if it should make a new user instance or just search from the ones already in data. ● Writes the data again and takes user to home page.

■ A bunch of mouse methods that we had to overwrite to include mouselistener, although their bodies are empty.

■ Void mousepressed(Mouse Event E)

- Again, goes through every page so each click coordinate conditional only applies when in the correct page.
- For each page, we calculated the coordinates of the buttons so that it accurately pressed them.
- The most complicated pages are 3 and 4, where similarly to the paint component, it uses conditionals to check if the machines are occupied, if they are pending, if so, since how long are they

ready. It also considers whether you are the owner of the load in or not. All of these conditionals change which methods are called when the button is pressed, as in empty or fill.

Amr, Javier, and Elena

- Other than that, for the queue page, it checks if you are already in the queue, as then it wont let you join the queue.

- Runner class (inside project)
  - Methods
    - Run method
      - It tries to receive a file from the server
      - Reads the data
      - repaints() → calls the paint component.
      - Tries to make the thread sleep.
- World:
  - Fields:
    - ArrayList<User> users:
      - Stores the address in the heap for all the users "registered" for the app
      - New people can register so the length must be flexible.
    - Washer [ ] washers
      - Stores the address in the heap for all the washers "registered" for the app
    - Dryer [ ] dryers
      - Stores the address in the heap for all the dryers "registered" for the app
    - User[] leaderboard
      - Stores the current users in order based on bubbles descending.
    - Queue wqueue
      - A queue for the washers
    - Queue dqueue
      - A queue for the dryers
  - Constructor (File data //which will come from IO redirection):

    // the constructor will read the data from the file and then create all the necessary users, washers, etc stored in the data
    Initializes the queues, the washer and dryer arrays with 3 of them each, initializes every individual machine and the user arraylist.

- Methods:
  - **setUpFile**
    - Void does not return anything
    - Takes in the file and makes a json object out of it. Assigns it to data.

  - readData
    - The data has dictionaries (key value pairs), arrays (lists) ● Void does not return but is used to initialize everything. ● Creates variables of json arrays or json objects for each entry in the json opening dictionary.
    - It initializes the machines, user arraylist and queues again. ● It creates an Iterator object that goes through every entry in each dictionary or array.
    - It compares the respective keys to the keys we expect, and takes their value and assigns it to a new variable of the needed type. ● It uses these variables to initialize the fields of all of the washers and users and link them together in the program based on the data.
    - The specific parsing techniques are explained in the code comments.
  - writeJson
    - Void, does not return
    - Creates a file with name ACRinse.txt
    - It makes a printwriter that uses this file.
    - It makes a json object.
    - It then makes a json array of users, each which is a LinkedHashMap of key value pairs, which are added based on the fields of each user instance. It also gets their status regarding a machine or active load.
    - It similarly makes a json array of machines, and makes a map for each one.
    - For the queues, it makes another map length 2 for the 2 queues and each holds the name of the queue and a json array as their value, holding the email of every user in each queue. Then, each array or map gets linked to the “node” parent to them, whether it is a dictionary or array.
    - It writes the json into the printwriter and tries to send file to server.
  - makeLeaderboard

        - Void does not return
        - Prints the leaderboard user array in world, with an index from 1 makes the leaderboard array in world of the people with the most bubbles.
        - Adds everyone in the arraylist to the leaderboard.
        - Calls merge sort to rank them based in bubbles.

Amr, Javier, and Elena

    - mergeSort
        - Does the merge sort algorithm by calling the compare bubbles method
    - checkWasherAvailability
        - Iterates through the washers and counts how many are available.
    - checkDrierAvailability
        - Same but with driers.
    - findUser(String email)
        - Returns a user
        - Goes through all the users and compares their email, then returns the correct user.

- User:
    - Fields:
        - **Private** Int Bubbles
            - Socialscore keeps track of how responsible user is with their loads /respecting the schedule
        - **Private** int index
            - Index of the user in the arraylist. -for data
        - **Private** String name
            - Differentiates user
        - Private String last name
        - **Private** String email
            - To make sure that each student has one profile/corresponds to one user
        - **Public** Load load
            - An instance of load class for each user (right now in the simplified version, each user can only have one load at a time)
        - **Public** overtimeLoads
            - Counts how many loads the user has taken out of a machine late.
        - **Public** totalLoads
            - Counts how many loads the user has completed overall.
    - Constructor (index, name, last name, email, bubbles, String load, String

machine type, String machineIndex, String washed, int totalLoads, int overtimeLoads)
        - (a lot of this arguments is become this is the version of the constructor that comes from the data)
        - Sets this.parameter = parameter
        - This.load = new load(load, machineindex, machinetype
        - Parses washed to a boolean and sets it.
    - Constructor ( same arguments until bubbles)

        - Calls userHelper method
        - All the load counters set to 0
    - Another overloaded constructor with same arguments as first except the machinetype and machineindex.
        - Calls userhelper
        - Deals with washed boolean and new load counters.
    - userHelper(takes in arguments)
        - Sets all arguments to their same parameters
        - New load(this)
    - Getter methods for name, last name, email, bubbles, index
    - Boolean equals
        - Compares the email of users, returns true if they are the same.
    - Int Comparebubbles (user)
        - Returns 1 if this user has greater bubbles then other
        - -1 otherwise
    - Void update bubbles (int change)
        - This.bubbles += change;
- Machine (abstract class)
    - Fields:
        - **Private** Boolean occupied
            - True when someone is using the machine
            - Private bc we don't want to inadvertently change it in another method
        - Public load, points to the load that is inside the machine
        - Public LocalDateTime end
            - Stores the time at which the current load finishes
        - **Private** Int timeremaining
            - Tracks time left in the cycle → updates every second

        - Private LocalDateTime pickUpEnd

        - Tracks the time after the machine finishes for the owner to pick it up before everyone else can take the load out too.
    - Public int index
        - Index in their respective arrays.
- Constructor ()
    - Occupied = false
    - Time remaining 0
    - Most of the code is in the super for the children classes.
- Methods
    - Getter method for occupied

    - **Public** void empty
        - Occupied false
        - Sets time remaining to 0
        - Sets machine index to -1 (not used)
        - Updates the load counters for the users.
        - Sets end to null again.
    - **Public** void Fill (load)
        - this .load=load
        - Sets occupied
    - Public void fill (end)
        - This method is only used by the data.
    - Int pickUpTimer
        - Calculates the time between the current time and the end of the time to pick it up for the owner after it is done
    - String getTimeRemaining
        - Returns time between current time and time for the laid to be done
    - String Time remainingToString
        - Calls timetostring
    - String pickUpTimeToString
        - Calls timetostring
    - String timetostring
        - Makes the strings pretty by making it a time format like 12:00
    - Void calculateBubbles
        - Calls the bubble change method in user depending on how late a user was to pick up their load.
        - Called when a machine is empty
        - Usually you get 100 bubbles, but you can lose 15, 100, 150, 300

- Washer extends Machine
    - Fields
        - **Private** Boolean extraRinse (if true then the total time would be 45) ■ Static int FINAL WASHINGtime = 35; // mins // 1 min in the program for testing
    - Constructor
        - super();
    - Methods
        - Isoccupied
            - Returns occupied
    - Empty
        - Super

Amr, Javier, and Elena

        - Sets this load to deadload.
        - Updates washing and washed variables
    - Fill (load)
        - Super fill (load)
        - Washing true
        - Sets up end and pickUpEnd for when the load finishes and for ten minutes after it finishes.
- Dryer extends Machine
    - Fields:
        - Static int FINAL time = 60; //mins // 2 for testing demo
    - Constructor
        - super();
    - Methods
        - Boolean isOccupied
        - 
        - Empty
            - Sets washing drying
            - Load to deadload.
        - Fill(load)
            - super(load)
            - Sets up end and pickupend
- Load: (instance of load = someone's clothes)
    - Fields:
        - **Public** User owner

- - - **Public** Boolean pending , true if they have overdue load
    - **Public** Boolean washing
    - **Public** Boolean drying
    - **Public** Boolean washed;
    - **Private** Boolean extraRinse;
  - Constructor
    - Constructor 1 (User u, String load, String machineType, String machineIndex)
      - This.user = u
      - This.pending = false
      - this.machineIndex = Integer.parseInt(machineIndex);
      - Washer If statement → if machine type is washer, washing is true, drying is false, washed is false, Project.World.washers[this.machineIndex].load = this;

Amr, Javier, and Elena

      - Dryer if statement → if machine type is dryer, drying is true, washing is false, washed is true, Project.World.washers[this.machineIndex].load = this;
      - Extrarinse = false
    - Constructor 2 ((User u)
      - This.owner = u
      - All other fields = false // machineindex = -1
  - Methods
    - Getter methods for all fields

In UserThreads.java we used code from:
https://heptadecane.medium.com/file-transfer-via-java-sockets-e8d4f30703a5

In Project.java we used code from:
https://www.geeksforgeeks.org/parse-json-java/