Matchwich

Software Engineering

ITSC-3155

Final Project Report

5/6/2020

Team 22

Sarah Stanish, Will Simpson, Daniel Issac, Paul John, Samuel Garn

Table of Contents

1. Introduction

1.1 Purpose

We are motivated to develop this application by negative experiences with other popular dating apps. When using other applications, it can be difficult to find a match that you are not only interested in, but also with whom you will have interesting conversations that will develop a relationship. This is primarily because matches are made solely based on physical appearance, rather than niche interests or facts about the user. A dating platform that gathers extensive information on such a mundane topic, sandwiches, brings out unique common interests between two people. These interests can extend far beyond sandwiches, through shared backgrounds, lifestyles, and other factors that influence their choices. This can lead to higher quality matches and provide members with a positive dating experience.

1.2 Scope

For this project, our team will be designing and developing a mobile application for single individuals called Matchwich. The goal of this software product is to provide user-driven, personalizable social matches for users on a real-time online platform. We will create this application in Android Studio and develop it in Java, a popular choice for Android apps. We will create a database to hold user information using MySQL and create the database in MySQLWorkbench. We will focus on keeping track of milestones and regularly reporting our progress to the stakeholders. We will develop features based on use cases and unit test each feature before releasing to our stakeholders. We will complete this project by the end of the semester and release the final product.

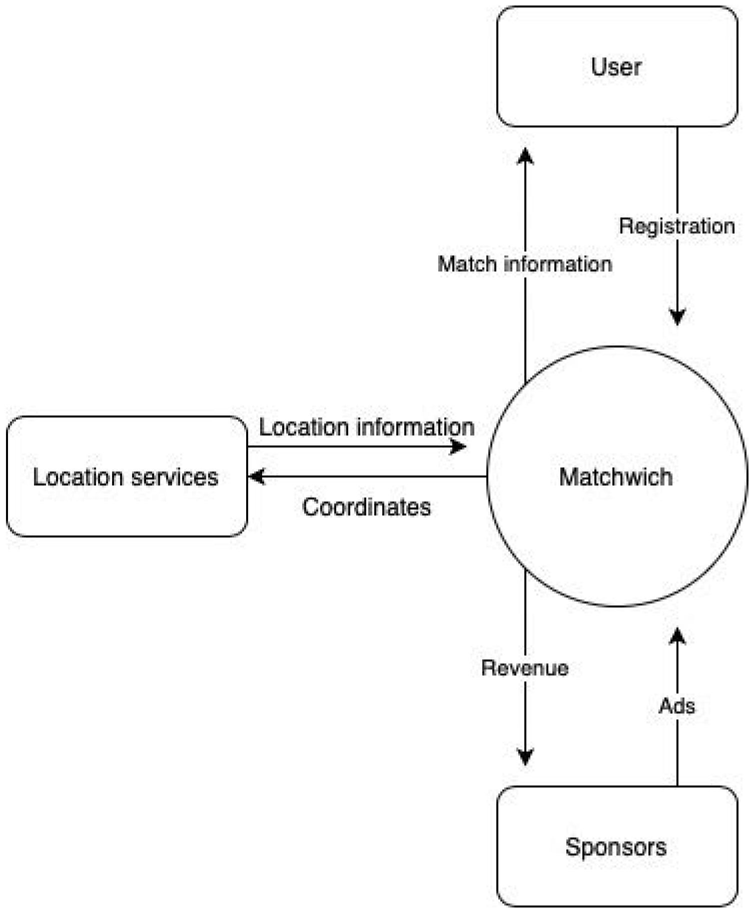
1.3 Definitions, Acronyms, and Abbreviations

- An **operating system (abbreviated OS)** is system software that manages computer software, hardware resources, and programs. Examples include Windows, Mac OS X, and Linux.
- An **application (often shortened to app)** is a program or group of programs designed for end users. Matchwich is a mobile application made for Android.
- **Android** is a mobile operating system designed for smartphones. It is based on a modified version of the Linux operating system. Android is developed primarily by Google and the Open Handset Alliance.
- A **database (commonly abbreviated DB)** is an electronically stored collection of data. For this project, we used a program called **MySQL** to create and maintain a database to store user records.
- **Java** is a platform-independent, high-level programming language. The Android operating system, which we used for this project, can use apps built in Java
- A **user interface** is a system that lets a user interact with a machine- in this case, an Android smartphone.
- A **parameter, referenced in code with the tag @param,** is an input given to part of a computer program. A **function (a small set of instructions)** inside a program may use several inputs from users or the system to accomplish various tasks or **return** an item or value.
- A **return value, referenced in code with the tag @return,** is output that a program's function gives when prompted. These functions may or may not take parameters; for example, an

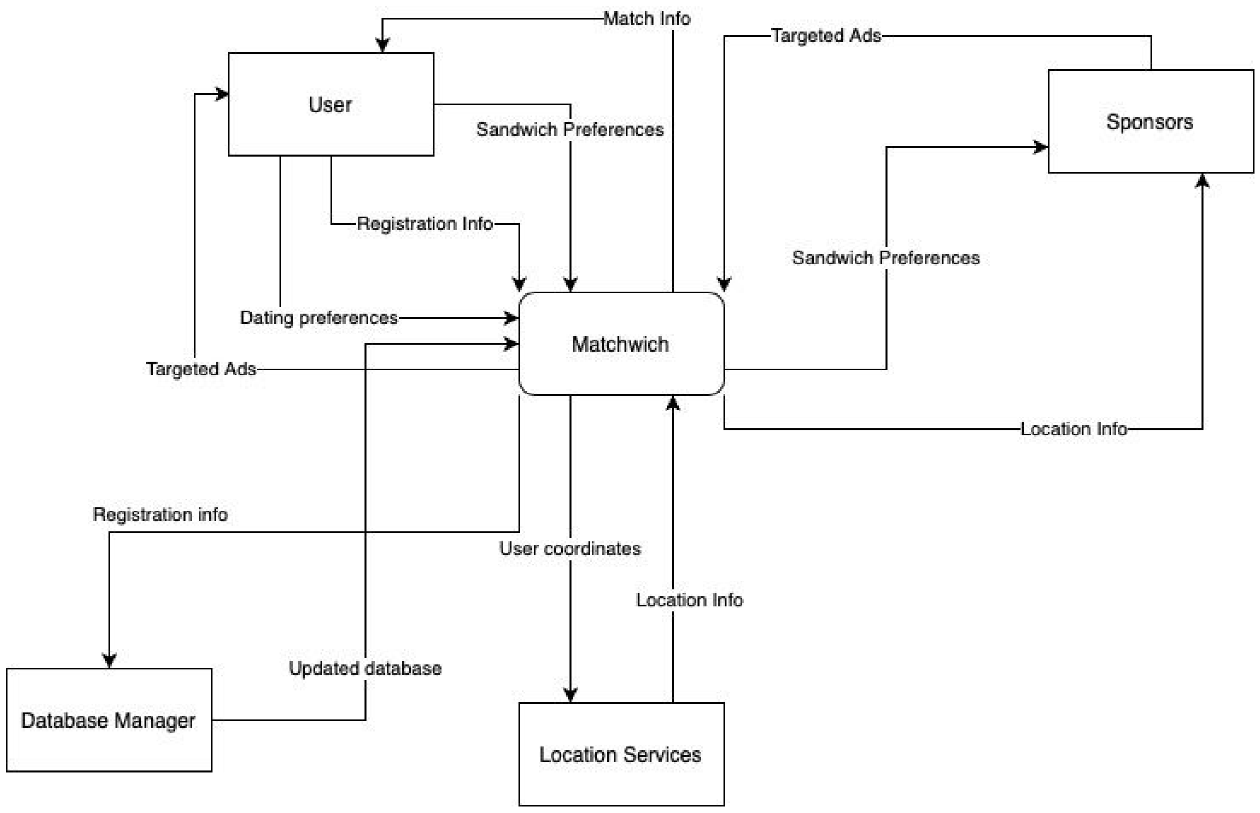
additional function would require two number parameters and “return” their sum as one number. A clock function might take no inputs and simply “return” the time and date when prompted to do so.

1.4 Context Diagram and Data Flow Diagram

Context Diagram



Data Flow Diagram



1.5 References

Stack Exchange Inc. 2020. *Stack Exchange*, 6 May 2020, stackexchange.com/.

Geo Data Source. 2020 GeoDataSource: *World Major Cities Database*, 6 May 2020, https://www.geodatasource.com/

1.6 Overview

Backend:
- MySQL Database: The database holds the users information and stores securely in the server

Frontend:
- Java with Android Studio: The frontend, made via a language and IDE designed for native Android application development, defines the UI and interactable components.

2. General Description

2.1 Project Perspective

Our team's project is a sandwich-based dating app called Matchwich™. Our team comes from a perspective that is mindful of the target market- socially gregarious people who have an interest in making new connections, and who are comfortable using online platforms for that purpose. We are also mindful of the competition in the dating app market from apps like Tinder, Hinge and Bumble.

2.2 Project Components

Our project consists of an Android frontend with a backend powered by a MySQL database and Java programming. The user interface was designed in Android Studio.

2.3 Project Goals

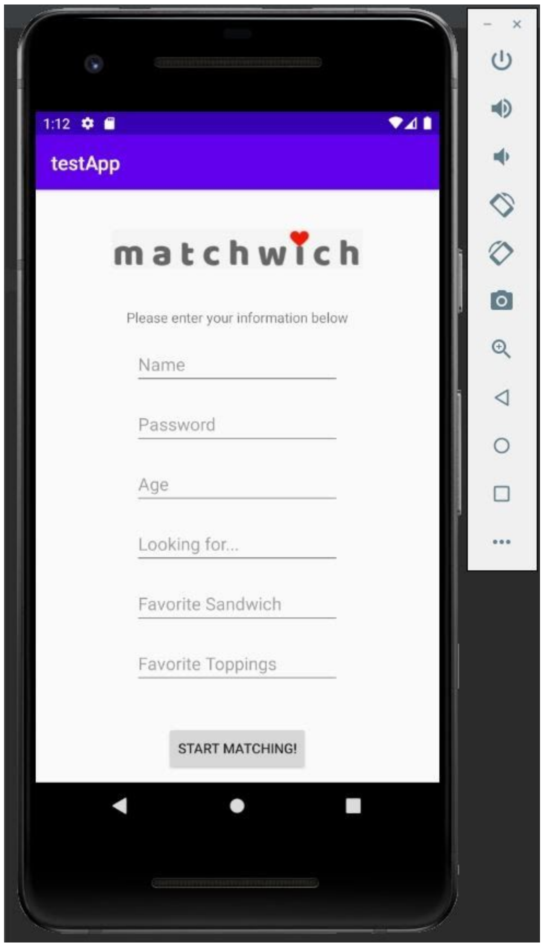
The goal of this application is to pair individuals together through a matching algorithm primarily based on their shared sandwich preferences, which is filled out through completing a form that is given upon account creation. This will give Matchwich users a fun and interesting commonality with their potential partner and serve as a convenient conversation starter. It is an exciting opportunity for users to find the love of their life or even a new friend by seeing someone's preferences in something as mundane as a sandwich. The sign up process will require new members to fill out a form to gather information about the member's opinion on various types of sandwiches, their favorite ingredients, and some other basic non-sandwich information. This will be used to seek out other members with similarities. Profiles will still have pictures and basic information, such as age, gender, and location.

3. Effort

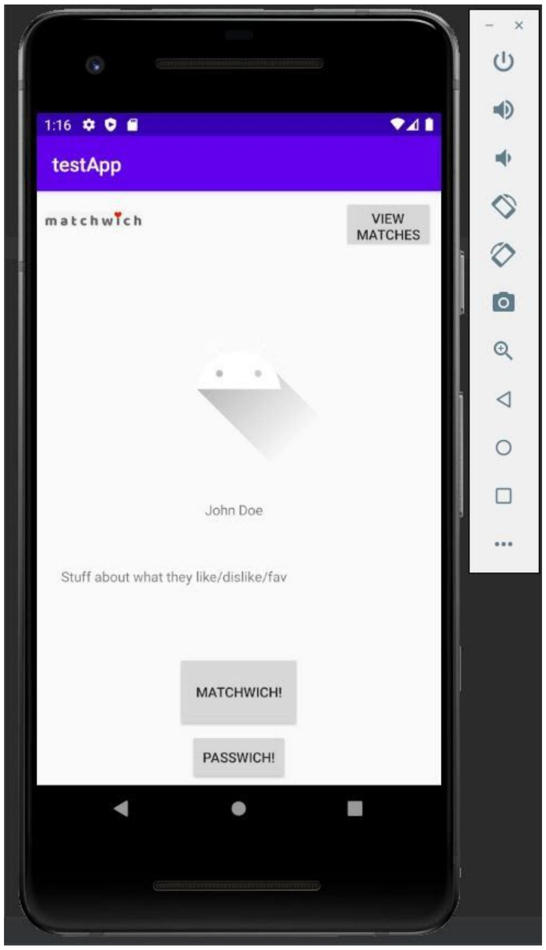
Task	Estimated Time of Research	Actual Time of Research	Estimated Coding Effort	Actual Coding Effort
Creating database in MySQLWorkbench	1 hour	2.5 hours	1 hour	1.5 hours
Designing and implementing matching algorithm	1 hour	2 hours	8 hours	10 hours
Designing the user interfaces	1 hour	3 hours	5 hours	8 hours

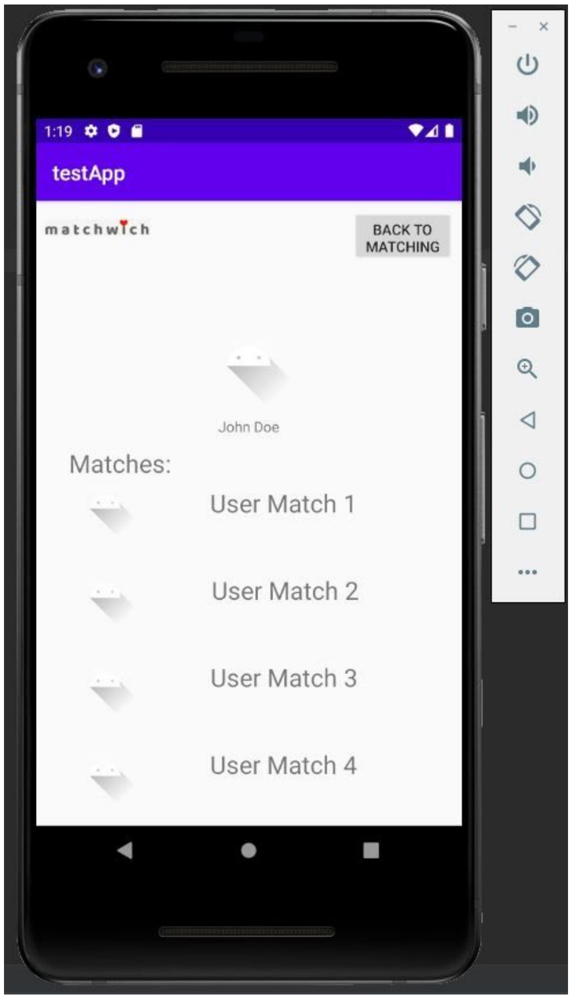
4. Programs Developed

4.1 User Interfaces:



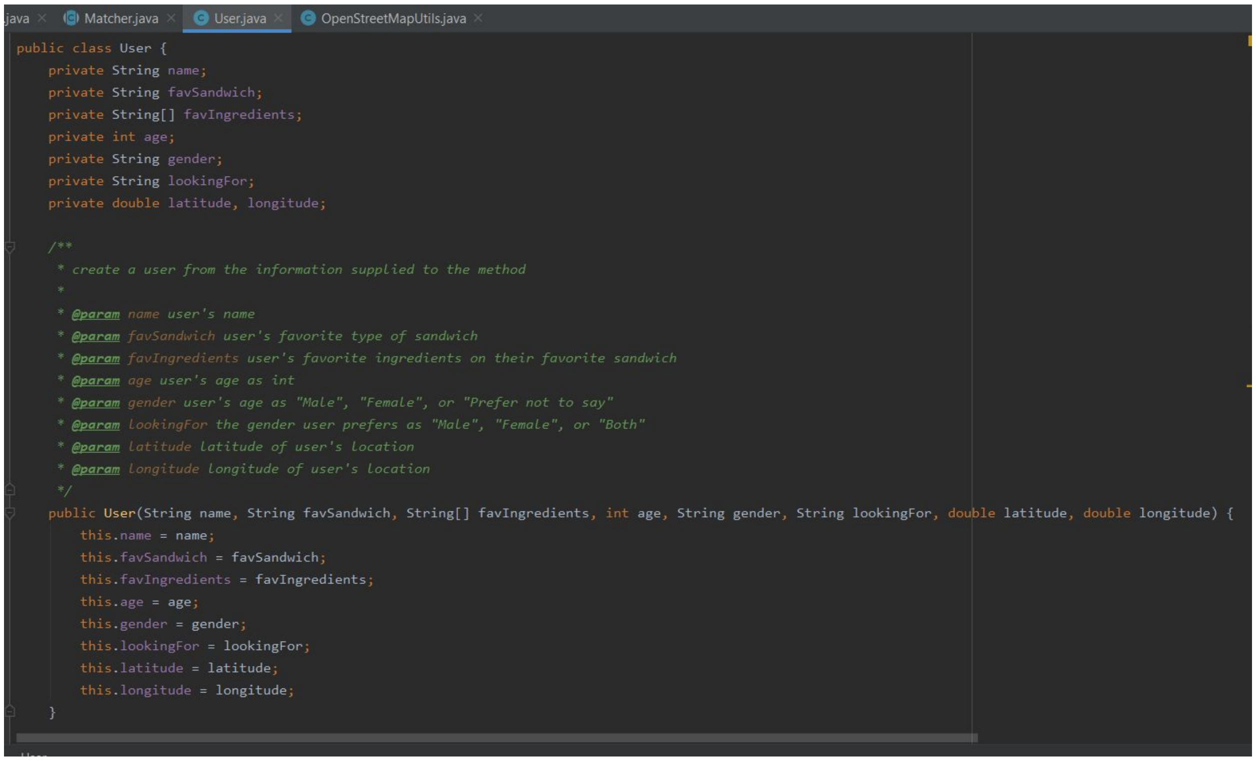
Matchwich


1:16
testApp
matchwich
VIEW MATCHES
John Doe
Stuff about what they like/dislike/fav
MATCHWICH!
PASSWICH!


1:19
testApp
matchwich
BACK TO MATCHING
John Doe
Matches:
User Match 1
User Match 2
User Match 3
User Match 4

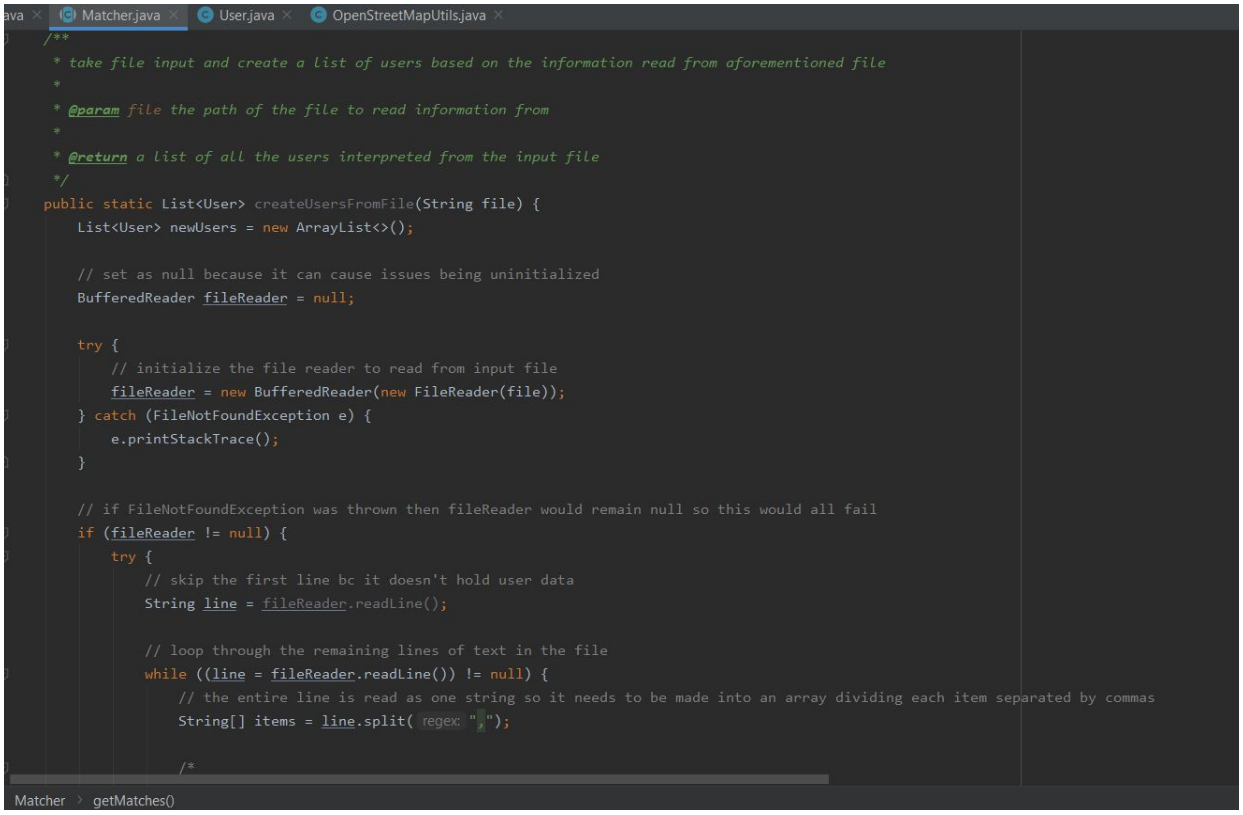
4.2 Matching algorithm:

User class

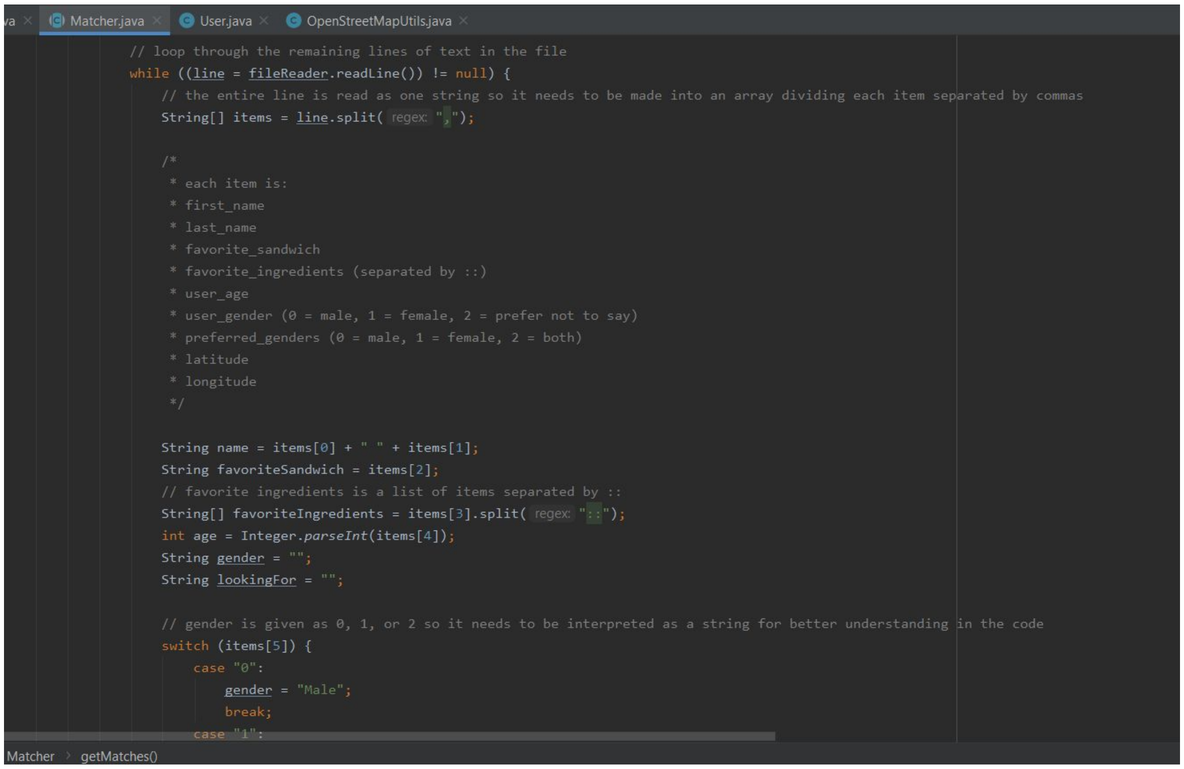


```java
public class User {
    private String name;
    private String favSandwich;
    private String[] favIngredients;
    private int age;
    private String gender;
    private String lookingFor;
    private double latitude, longitude;

    /**
     * create a user from the information supplied to the method
     *
     * @param name user's name
     * @param favSandwich user's favorite type of sandwich
     * @param favIngredients user's favorite ingredients on their favorite sandwich
     * @param age user's age as int
     * @param gender user's age as "Male", "Female", or "Prefer not to say"
     * @param lookingFor the gender user prefers as "Male", "Female", or "Both"
     * @param latitude latitude of user's location
     * @param longitude longitude of user's location
     */
    public User(String name, String favSandwich, String[] favIngredients, int age, String gender, String lookingFor, double latitude, double longitude) {
        this.name = name;
        this.favSandwich = favSandwich;
        this.favIngredients = favIngredients;
        this.age = age;
        this.gender = gender;
        this.lookingFor = lookingFor;
        this.latitude = latitude;
        this.longitude = longitude;
    }
```

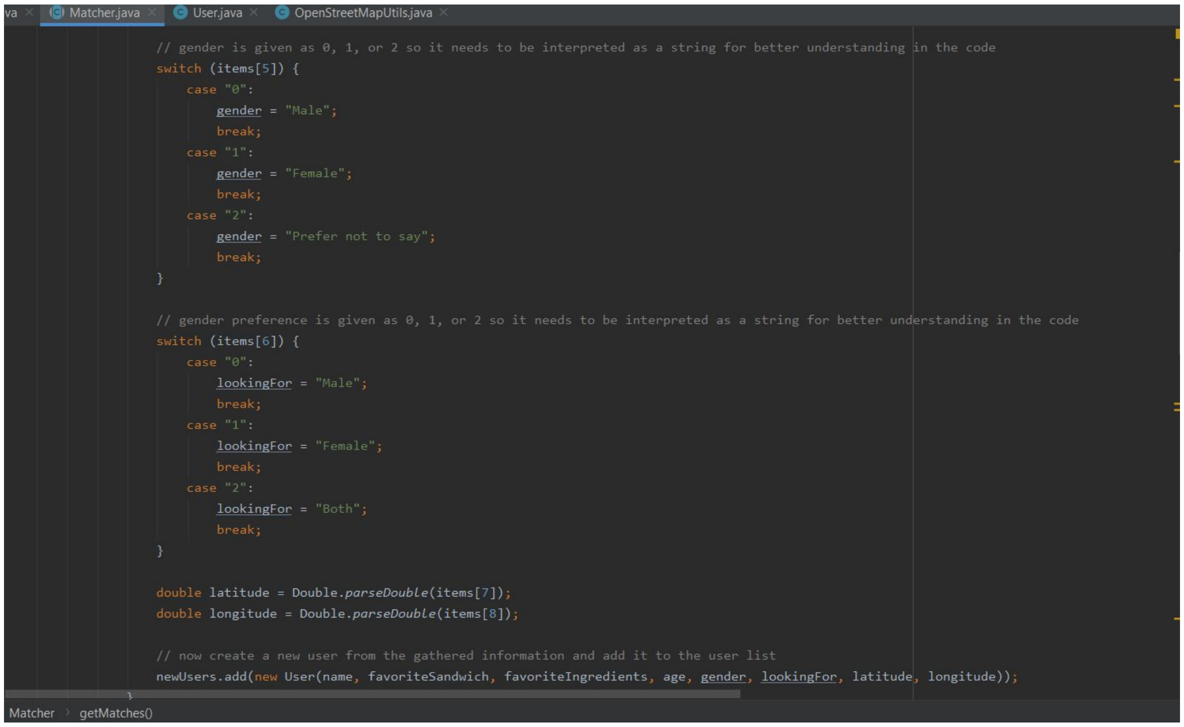
`createUsers()` method



```
/**
 * take file input and create a list of users based on the information read from aforementioned file
 *
 * @param file the path of the file to read information from
 *
 * @return a list of all the users interpreted from the input file
 */
public static List<User> createUsersFromFile(String file) {
    List<User> newUsers = new ArrayList<>();

    // set as null because it can cause issues being uninitialized
    BufferedReader fileReader = null;

    try {
        // initialize the file reader to read from input file
        fileReader = new BufferedReader(new FileReader(file));
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }

    // if FileNotFoundException was thrown then fileReader would remain null so this would all fail
    if (fileReader != null) {
        try {
            // skip the first line bc it doesn't hold user data
            String line = fileReader.readLine();

            // loop through the remaining lines of text in the file
            while ((line = fileReader.readLine()) != null) {
                // the entire line is read as one string so it needs to be made into an array dividing each item separated by commas
                String[] items = line.split( regex: ",");

                /*
```

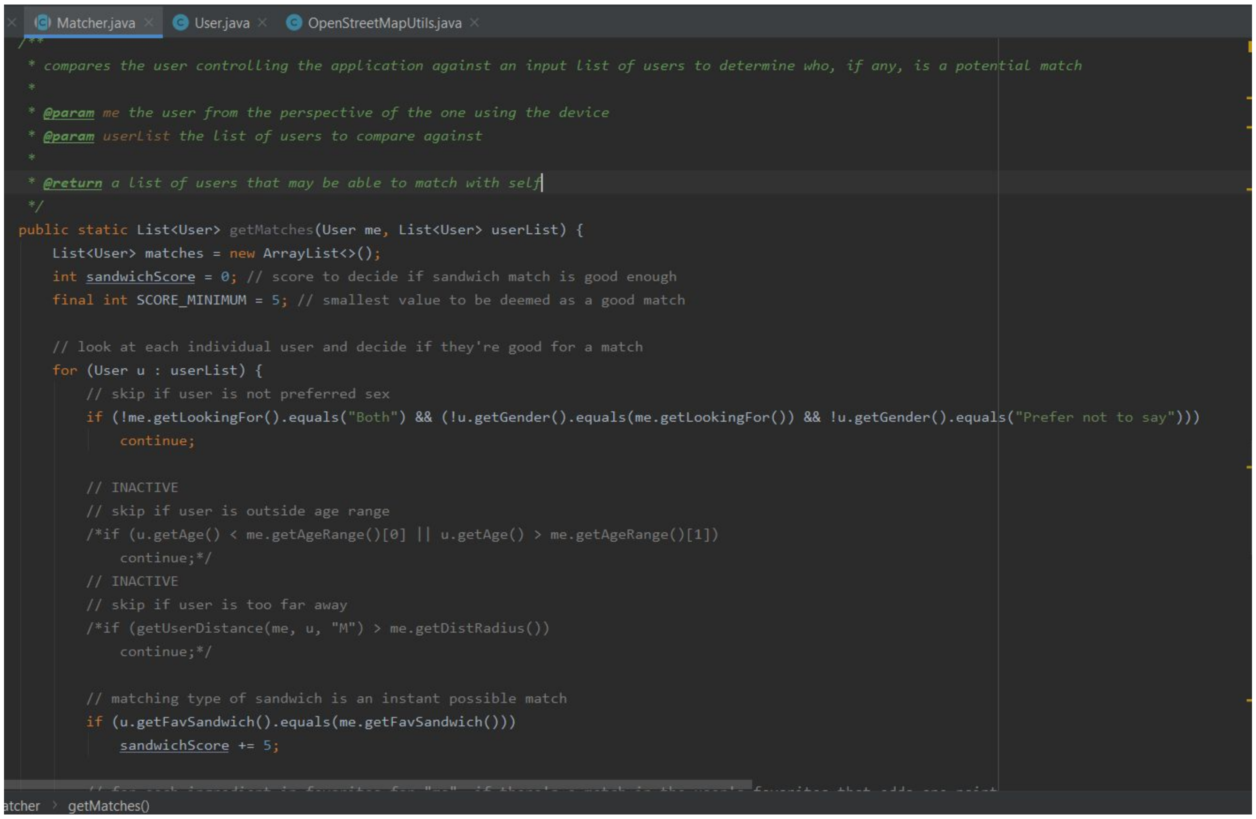


```
        // loop through the remaining lines of text in the file
        while ((line = fileReader.readLine()) != null) {
            // the entire line is read as one string so it needs to be made into an array dividing each item separated by commas
            String[] items = line.split( regex: ",");

            /*
             * each item is:
             * first_name
             * last_name
             * favorite_sandwich
             * favorite_ingredients (separated by ::)
             * user_age
             * user_gender (0 = male, 1 = female, 2 = prefer not to say)
             * preferred_genders (0 = male, 1 = female, 2 = both)
             * latitude
             * longitude
             */

            String name = items[0] + " " + items[1];
            String favoriteSandwich = items[2];
            // favorite ingredients is a list of items separated by ::
            String[] favoriteIngredients = items[3].split( regex: "::");
            int age = Integer.parseInt(items[4]);
            String gender = "";
            String lookingFor = "";

            // gender is given as 0, 1, or 2 so it needs to be interpreted as a string for better understanding in the code
            switch (items[5]) {
                case "0":
                    gender = "Male";
                    break;
                case "1":
```

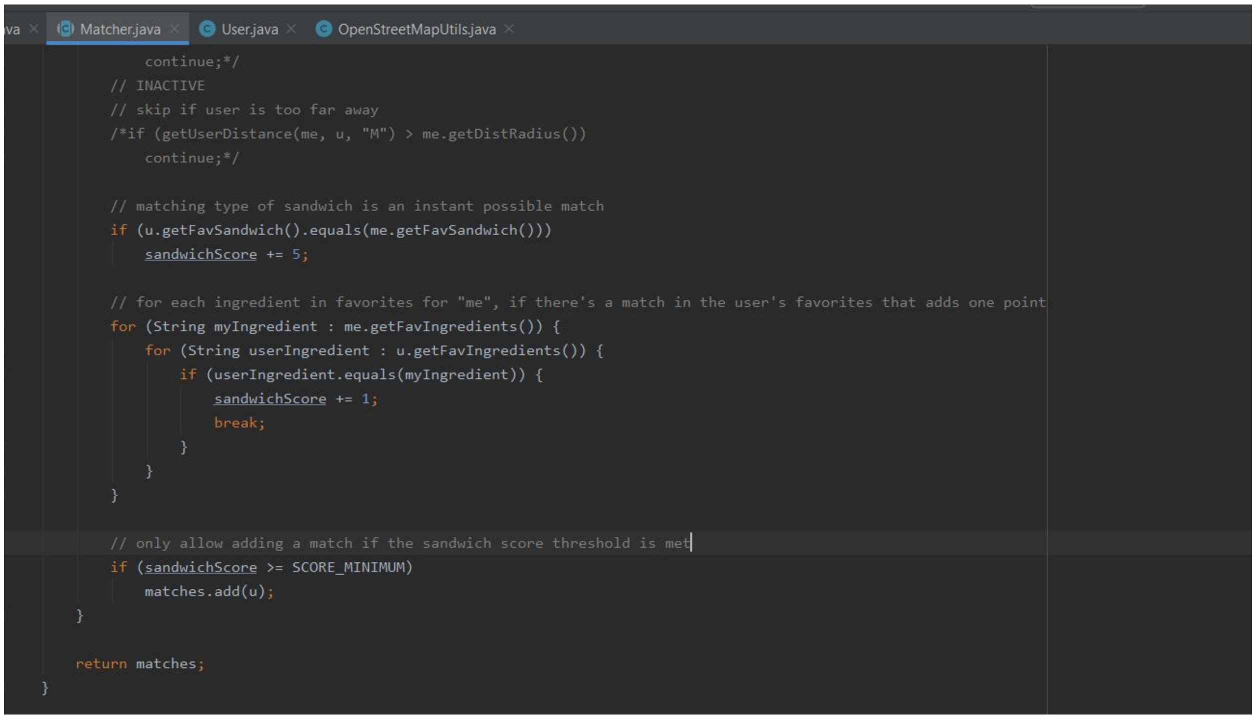


```
            // gender is given as 0, 1, or 2 so it needs to be interpreted as a string for better understanding in the code
            switch (items[5]) {
                case "0":
                    gender = "Male";
                    break;
                case "1":
                    gender = "Female";
                    break;
                case "2":
                    gender = "Prefer not to say";
                    break;
            }

            // gender preference is given as 0, 1, or 2 so it needs to be interpreted as a string for better understanding in the code
            switch (items[6]) {
                case "0":
                    lookingFor = "Male";
                    break;
                case "1":
                    lookingFor = "Female";
                    break;
                case "2":
                    lookingFor = "Both";
                    break;
            }

            double latitude = Double.parseDouble(items[7]);
            double longitude = Double.parseDouble(items[8]);

            // now create a new user from the gathered information and add it to the user list
            newUsers.add(new User(name, favoriteSandwich, favoriteIngredients, age, gender, lookingFor, latitude, longitude));
        }
```

`getMatches()` method



```java
/**
 * compares the user controlling the application against an input list of users to determine who, if any, is a potential match
 *
 * @param me the user from the perspective of the one using the device
 * @param userList the list of users to compare against
 *
 * @return a list of users that may be able to match with self
 */
public static List<User> getMatches(User me, List<User> userList) {
    List<User> matches = new ArrayList<>();
    int sandwichScore = 0; // score to decide if sandwich match is good enough
    final int SCORE_MINIMUM = 5; // smallest value to be deemed as a good match

    // look at each individual user and decide if they're good for a match
    for (User u : userList) {
        // skip if user is not preferred sex
        if (!me.getLookingFor().equals("Both") && (!u.getGender().equals(me.getLookingFor()) && !u.getGender().equals("Prefer not to say")))
            continue;

        // INACTIVE
        // skip if user is outside age range
        /*if (u.getAge() < me.getAgeRange()[0] || u.getAge() > me.getAgeRange()[1])
            continue;*/
        // INACTIVE
        // skip if user is too far away
        /*if (getUserDistance(me, u, "M") > me.getDistRadius())
            continue;*/

        // matching type of sandwich is an instant possible match
        if (u.getFavSandwich().equals(me.getFavSandwich()))
            sandwichScore += 5;

        // for each ingredient in favorites for "me", if there's a match in the user's favorites that adds one point
```

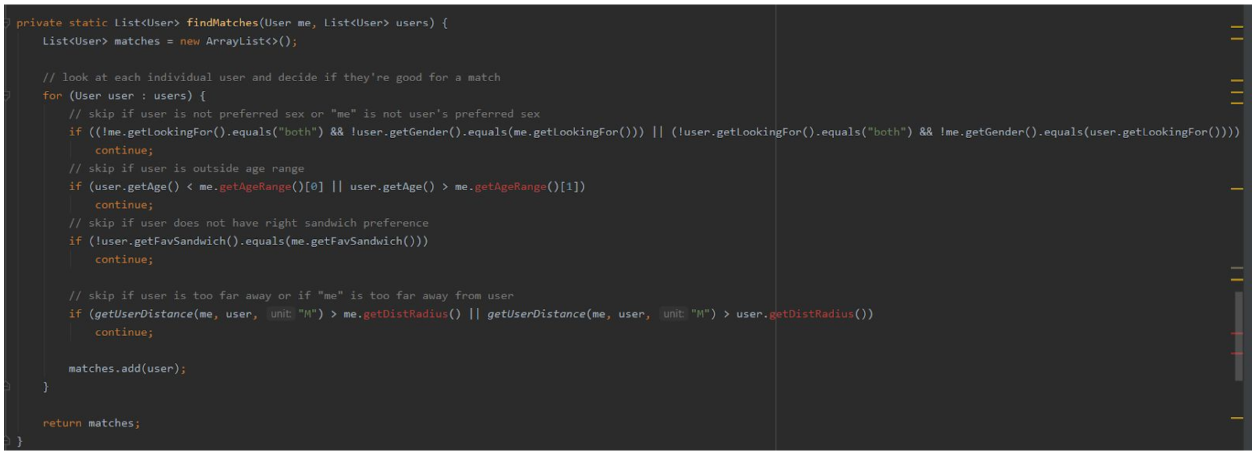


```java
            continue;*/
        // INACTIVE
        // skip if user is too far away
        /*if (getUserDistance(me, u, "M") > me.getDistRadius())
            continue;*/

        // matching type of sandwich is an instant possible match
        if (u.getFavSandwich().equals(me.getFavSandwich()))
            sandwichScore += 5;

        // for each ingredient in favorites for "me", if there's a match in the user's favorites that adds one point
        for (String myIngredient : me.getFavIngredients()) {
            for (String userIngredient : u.getFavIngredients()) {
                if (userIngredient.equals(myIngredient)) {
                    sandwichScore += 1;
                    break;
                }
            }
        }

        // only allow adding a match if the sandwich score threshold is met
        if (sandwichScore >= SCORE_MINIMUM)
            matches.add(u);
    }

    return matches;
}
```

4.3 Comparison:

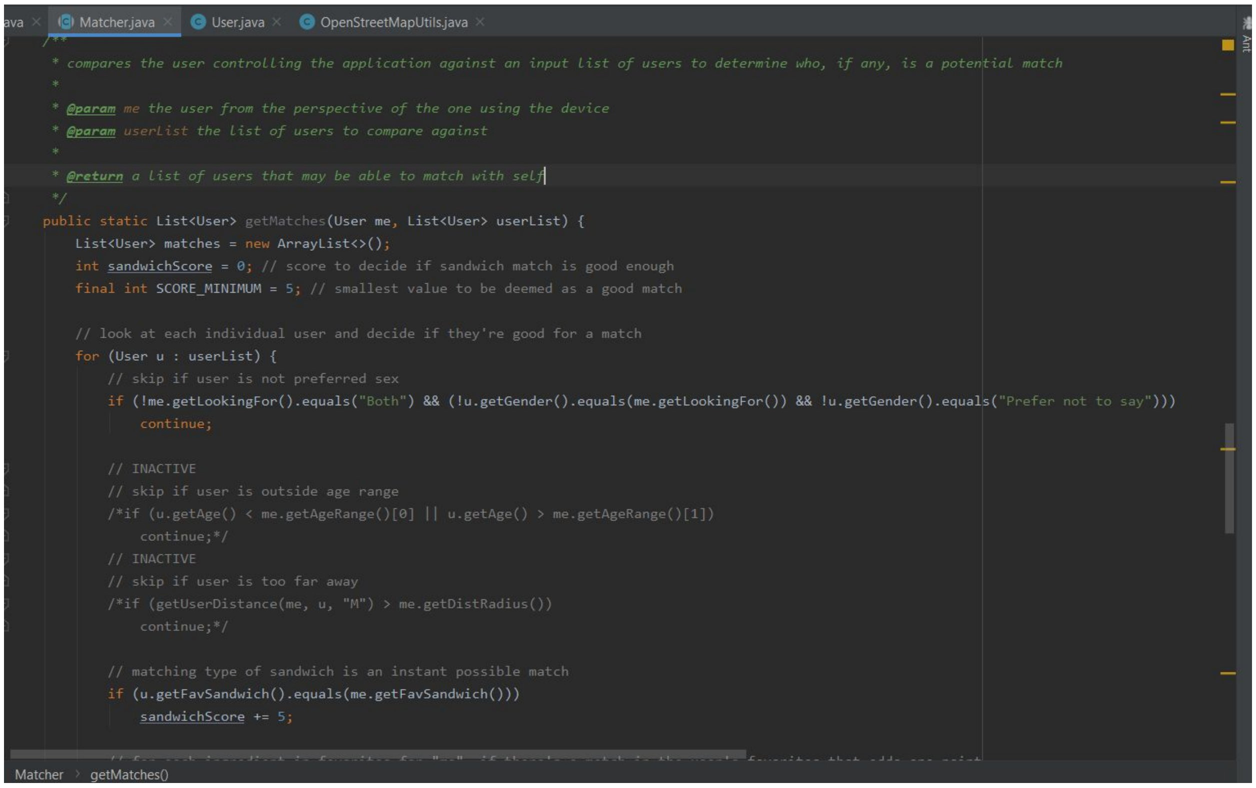
Matching algorithm

Old

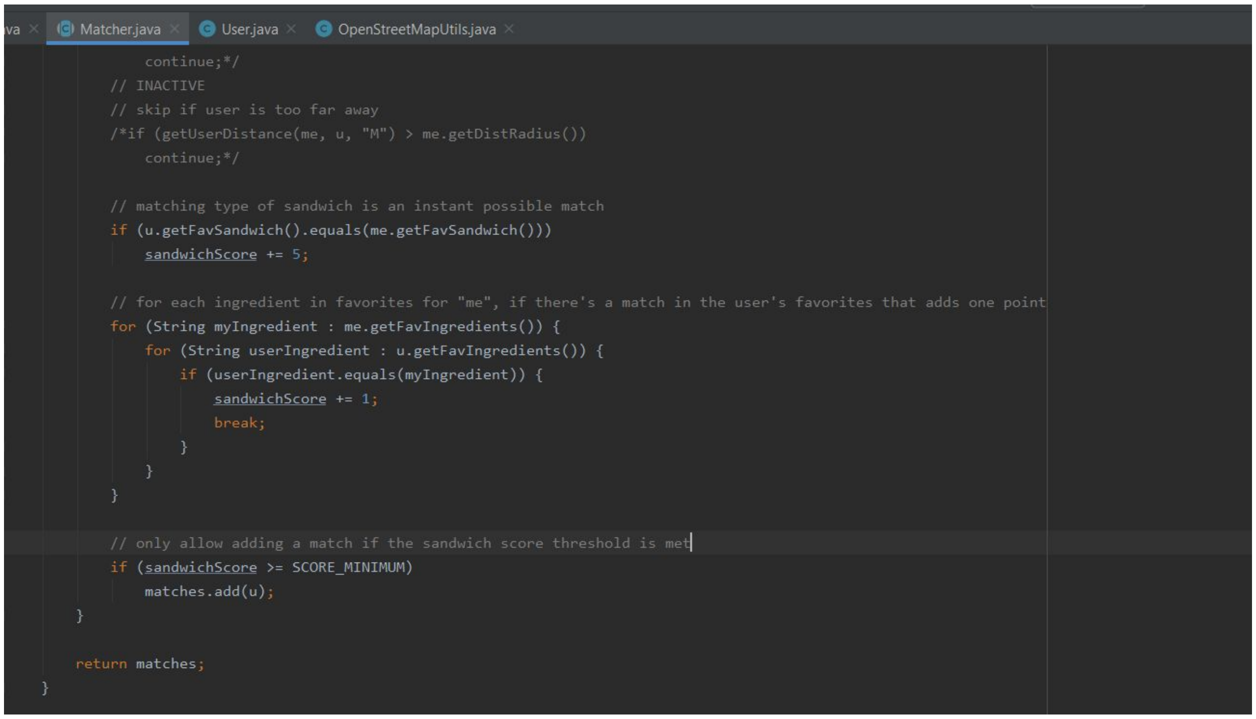


```java
private static List<User> findMatches(User me, List<User> users) {
    List<User> matches = new ArrayList<>();

    // look at each individual user and decide if they're good for a match
    for (User user : users) {
        // skip if user is not preferred sex or "me" is not user's preferred sex
        if ((!me.getLookingFor().equals("both") && !user.getGender().equals(me.getLookingFor())) || (!user.getLookingFor().equals("both") && !me.getGender().equals(user.getLookingFor())))
            continue;
        // skip if user is outside age range
        if (user.getAge() < me.getAgeRange()[0] || user.getAge() > me.getAgeRange()[1])
            continue;
        // skip if user does not have right sandwich preference
        if (!user.getFavSandwich().equals(me.getFavSandwich()))
            continue;

        // skip if user is too far away or if "me" is too far away from user
        if (getUserDistance(me, user, unit: "M") > me.getDistRadius() || getUserDistance(me, user, unit: "M") > user.getDistRadius())
            continue;

        matches.add(user);
    }

    return matches;
}
```

New



```java
/**
 * compares the user controlling the application against an input list of users to determine who, if any, is a potential match
 *
 * @param me the user from the perspective of the one using the device
 * @param userList the list of users to compare against
 *
 * @return a list of users that may be able to match with self
 */
public static List<User> getMatches(User me, List<User> userList) {
    List<User> matches = new ArrayList<>();
    int sandwichScore = 0; // score to decide if sandwich match is good enough
    final int SCORE_MINIMUM = 5; // smallest value to be deemed as a good match

    // look at each individual user and decide if they're good for a match
    for (User u : userList) {
        // skip if user is not preferred sex
        if (!me.getLookingFor().equals("Both") && (!u.getGender().equals(me.getLookingFor()) && !u.getGender().equals("Prefer not to say")))
            continue;

        // INACTIVE
        // skip if user is outside age range
        /*if (u.getAge() < me.getAgeRange()[0] || u.getAge() > me.getAgeRange()[1])
            continue;*/
        // INACTIVE
        // skip if user is too far away
        /*if (getUserDistance(me, u, "M") > me.getDistRadius())
            continue;*/

        // matching type of sandwich is an instant possible match
        if (u.getFavSandwich().equals(me.getFavSandwich()))
            sandwichScore += 5;
```



```
            continue;*/
        // INACTIVE
        // skip if user is too far away
        /*if (getUserDistance(me, u, "M") > me.getDistRadius())
            continue;*/

        // matching type of sandwich is an instant possible match
        if (u.getFavSandwich().equals(me.getFavSandwich()))
            sandwichScore += 5;

        // for each ingredient in favorites for "me", if there's a match in the user's favorites that adds one point
        for (String myIngredient : me.getFavIngredients()) {
            for (String userIngredient : u.getFavIngredients()) {
                if (userIngredient.equals(myIngredient)) {
                    sandwichScore += 1;
                    break;
                }
            }
        }

        // only allow adding a match if the sandwich score threshold is met
        if (sandwichScore >= SCORE_MINIMUM)
            matches.add(u);
    }

    return matches;
}
```

Between the old and new versions of the methods, it's immediately evident the amount of code is different, in that the new version is quite longer. This is because of adding more items to compare against and another way to determine a match, being the favorite ingredients list of each user and the sandwich score respectively. The new matching algorithm is more complex and intensive to fulfill the goal of the matching algorithm more accurately and effectively.

5. Discussions and Conclusions

5.1 What can be concluded:

One of the main challenges that we faced as a team is being able to work at a similar pace. We each took on different roles and we had to learn how to work together effectively and schedule out our time. The biggest help in this challenge was the use of team meetings. Having weekly team meetings and using the project tracker and scheduling methods helped us to develop a workflow and keep track of our progress. This is something we did not realize was important until starting our project. Team meetings improved accountability and helped us focus on our common goal.

During this project, we also discovered the importance of documenting our work. We did this by using DFDs and context diagrams to plan our project, as well as using GitHub to collaborate on our project and update the group when we made changes. Planning and collaboration are important in the industry and it was helpful to gain some experience with these things now. We did not anticipate the challenge of trying to combine all of the work that we did separately, so it is good that we had the tools that we did to make the project work.

The final challenge that we faced was time estimation. We did not think that it would take us a long time to complete certain tasks, like the frontend portion, yet the frontend took the longest amount of time because there were more details than we anticipated. These are things that we will continue to improve on in the future, but this course was a learning experience for us and we gained valuable knowledge and experience.