What design patterns/concepts I used:

1. Liskov Substitution principal
    - There are separate classes made for matrices. (Check src/matrices)
        - Matrix<E>
            - SquareMatrix<E, T>
            - Fmatrix<E>
    - Instead of having a Cfiletering object that does it all another class was implemented (check src/userData)
        - UserData<E, T> class
            - Prints data (using overriden toString() method)
            - Is able to find user pairs and print them out.
2. Generics design pattern
    - Used in all classes except for CfilteringDriver
3. Single Responsibility
    - The way the classes are split up allow for this.
4. Iterator Design pattern
    - Included in Matrix<E>. Check src/matrices/Matrix.java
5. Interfaces
    - SquareMatrix<E, T> implements AddByMatrixMultiply<T>. Check src/matrices/AddByMatrixMultiply.java
        - Has methods addByMatrixMultiply() and takeNDiffSquare()
    - Fmatrix<E> implements AddByFile<E>. Check src/matricies/AddByFile.java
        - Has methods addByFile()
6. Exceptions
    - userData constructor throws FileNotFoundException.
        - FileName provided from user not found.
    - FMatrix uses try-catch while reading opening file.
        - Return null if any type of error occurs. This is due to the only error being that the file was not found and or file not formatted correctly.
    - FMatrix uses try-catch while populating 2d-array/matrix.
    - Matrix uses try-catch for iterator.hasNext().
        - Checks if the ‘next var’ is possible
    - CfilteringDriver uses try-catch for taking in userInput.
    - CfilteringDriver uses try-catch for runProg()
        - If there is an error while constructing UserData object then the errors are caught.