# 7 Two Dimensional Arrays: Biometrics Activity

Name:
Section:

## Introduction: Biometrics Technology

You know that cool way your phone can recognize your fingerprint or face to unlock it? Well, that's thanks to something called *biometric* authentication. Biometrics is all about studying unique characteristics about an individual, such as the fingerprint pattern or the way someone's face looks, to make sure it's really that person.

Think of it like a *secret handshake* between that individual and their device. This technology is a big deal in today's gadgets and apps because it helps make things super safe, makes the interactions smoother, and helps keep personal information private by making sure only that individual can get in.

The task at hand involves understanding the way biometric information gets stored on a device. During this activity, you will explore the following:

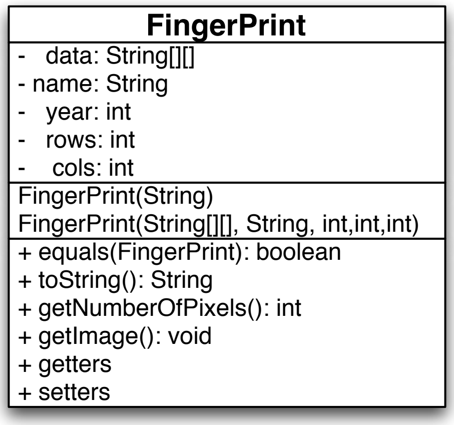
1. Discover the concept of biometrics and their significance in safeguarding data.

2. Gain insight into the process of extracting biometric data and organizing it within a data structure. This enables computer users to confirm their identity whenever they need to access information.

3. Learn about statistical errors and how to arrive at a solution using a percentage-based approach.

## Part I. Programming: File Processing and Two-dimensional Arrays

Create an object (`FingerPrint.java`) according to the characteristics.



```
FingerPrint
-   data: String[][]
- name: String
-   year: int
-   rows: int
-    cols: int
FingerPrint(String)
FingerPrint(String[][], String, int,int,int)
+ equals(FingerPrint): boolean
+ toString(): String
+ getNumberOfPixels(): int
+ getImage(): void
+ getters
+ setters
```

## **FingerPrint Object Implementation:** `FingerPrint.java`

A `FingerPrint` object consists the following properties

- data: a 2-D array of `Strings`.
- name: a `String` that is assigned to this fingerprint
- year: an `int`, representing the year that this fingerprint was registered
- rows: an `int`, representing the number rows of the image
- cols: an `int`, representing the number of columns of the image

The `FingerPrint` object owns the following methods:

1. *Implement* a constructor that takes a `String`, representing the file name where the information of the `FingerPrint` is located. This constructor will extract the information from the file and will initialize the fields from this `FingerPrint` Object.
2. *Implement* a customized constructor that will take all needed parameters to initialize the fields from this `FingerPrint` Object.
3. *Implement* a method named `getNumberOfPixels`. This method shall return a number representing the number of pixels of this `Fingerprint`, i.e., number of rows $\times$ cols
4. *Implement* a method named `toString()`. This method provides a `String` representation of this `Fingerprint`, i.e.,

   ```
   Fingerprint for:  <name>.  Year Registered:  <year>.  Number
   of Pixeles:  <pixels>
   ```

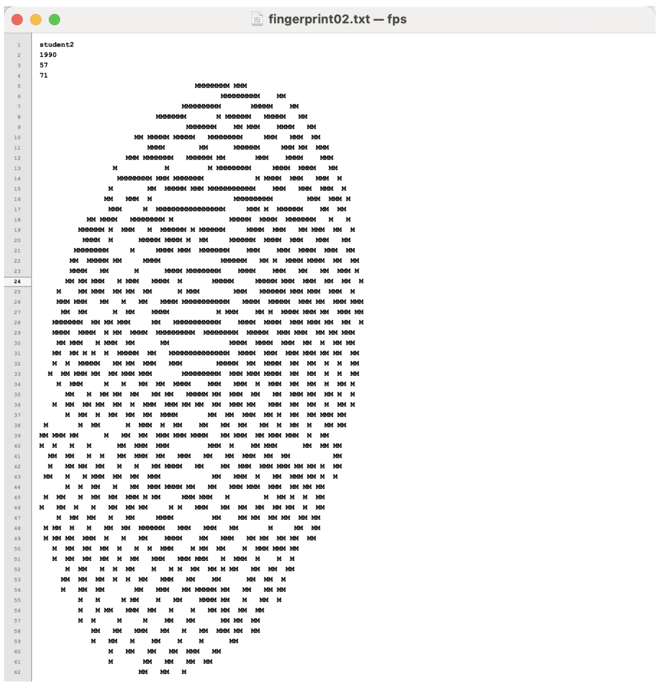
5. *Implement* a method named `getImage()`. This method will print the 2-D array containing the data of the figure corresponding from the file loaded. This method shall print the textual representation of the fingerprint as shown in **File Processing** Section.
6. *Implement* a method named `equals` which accepts a `FingerPrint` object as an argument. This function should return to `true` if the provided fingerprint matches the current `Fingerprint` in this object. To determine a match, compare the *name*, *year*, number of *rows*, and *columns* with the argument's fingerprint. Additionally, ensure that each character in the two-dimensional array corresponds with the character in the current object's array.

*Test* each object by reading information from a file, and create instances of the object.

## **File Processing:** `FileProcessor.java`

A Fingerprint object is configured in a text file, where the first 4 lines of the file provide information about the fingerprint object.

- student2: corresponds to the *name* assigned to the fingerprint
- 1990: corresponds to the *year* assigned to the fingerprint
- 58: corresponds to the rows, and
- 72: corresponds to the columns.



```
student2
1990
57
71
```

Figure 3: Fingerprint Representation. Human fingerprint (left) and a textual representation (right) stored in a file

There are several files located at the repository (check the QR-code for more information). This is the example of `fingerprint02.txt`

The remaining lines of characters corresponds to the fingerprint image. For example,

In `FileProcessor.java`:

- *Generate* a global instance of the Fingerprint object named *original*, which will represent the user's authentic fingerprint. This instance is intended for comparison with other fingerprints during the simulation of access control, serving as the benchmark for the user's fingerprint.

- *Define* an int variable named `maxTries` to denote the maximum allowable attempts for accessing a system before triggering a lock due to incorrect responses.

- *Implement* a method named `loadFingerPrint()`. which accepts a String parameter named `fileName` to denote the name of the file containing fingerprint data. This function is responsible for reading and interpreting the file according to the specified attributes. Ultimately, the function should produce and return an instance of a Fingerprint object, constructed from the data extracted from the file.

- *Generate* three different instances of the `FingerPrint` objects.

- *Code* the lines that are in charge of adding the three instances of `FingerPrints` into an array of *type* `FingerPrint`

- *Test* the methods for each instance. Use model 1 depicted in Figure 4

## Part II. Cyber-awareness: Authentication and Validation

To safeguard access control against unauthorized intrusions, various protective measures are in place, including protocols that deter repeated attempts to breach systems. One such mechanism is referred to as *incident response*. Incident Response (IR) entails a series of procedures

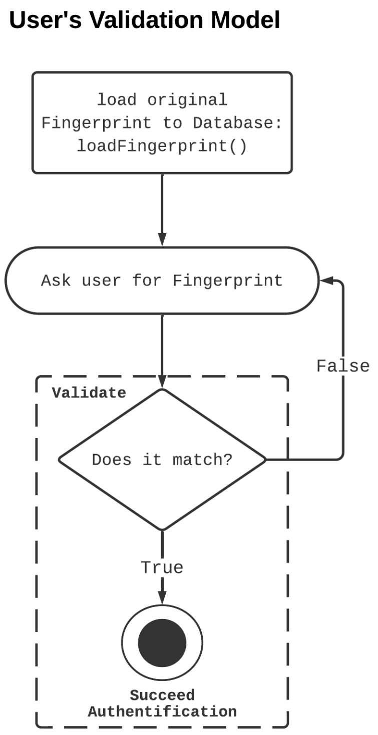


Figure 4: Model 1 – Validation of a fingerprint based on comparing a fingerprint against an original one

aimed at preparing for, detecting, containing, and recovering from a data breach. In scenarios involving biometric authentication, a predefined limit exists for the number of access attempts allowed before a specific *strategy* is enacted. Such a strategy could involve actions like device blocking, notification to an authority, account deactivation, capturing an image of the intruder, and transmitting it to a designated contact. In this section of the task, the objective is to establish an incident response mechanism in line with the representation depicted in Figure 5. This entails the following steps:

- *Define* a global variable named *maxTries*, dictating the permissible number of attempts a user can make to access the system without incurring penalties according to the incident response strategy.

- *Define* a variable named *tries* to maintain a count of the unsuccessful attempts made to gain access.

- *Implement* a mechanism that enables user interaction with the system, validating the fingerprint provided, provided that the number of attempts does not surpass *maxTries* and the presented fingerprint matches the original one.

By carrying out the aforementioned steps, the system can establish an effective incident response protocol as outlined in the provided diagram, offering protection against unauthorized access attempts.

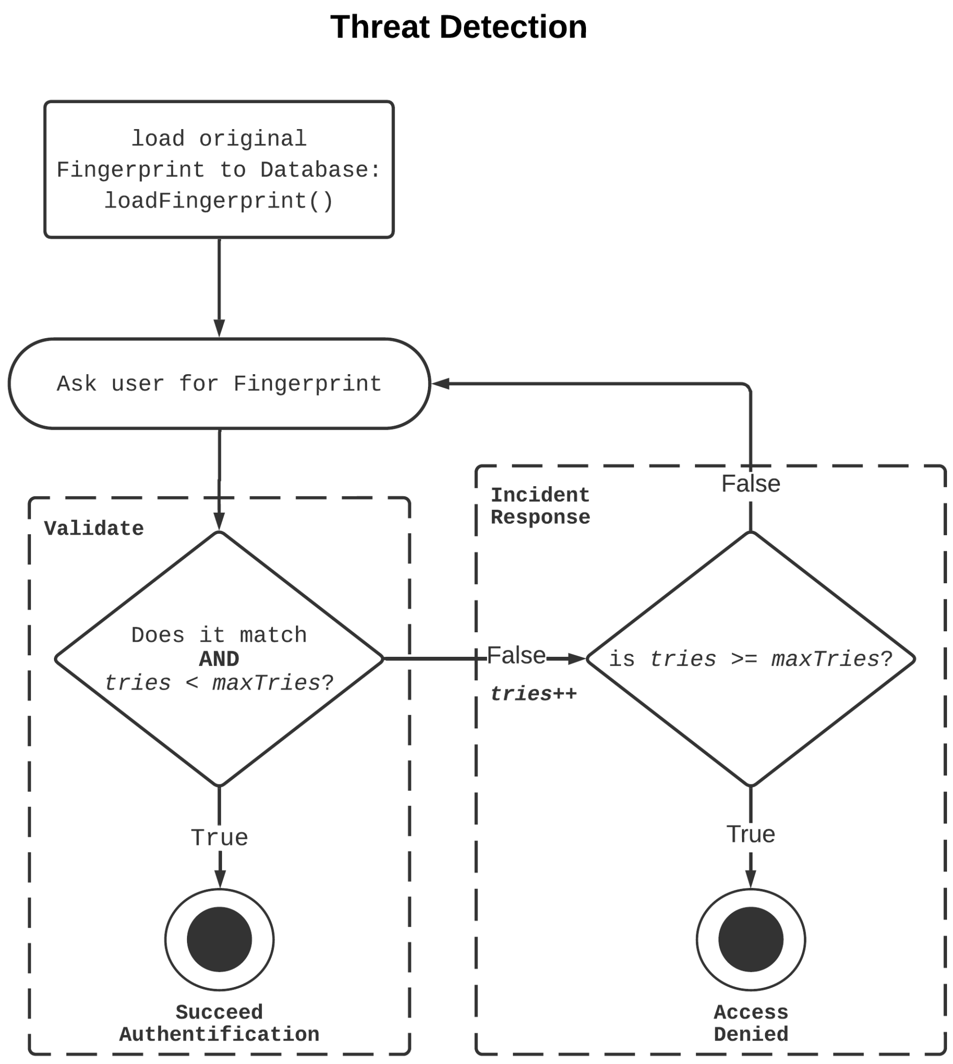


Figure 5: Model 2 – Utilizing Incident Response for a Biometric Authentication Failure

## Part III. Data and Statistical Analytics

From time to time, it becomes essential to incorporate adaptability within a system to be “useful” to typical situations in real life. For instance, there are situations where the biometric authentication system might experience partial failures due to imperfections in the provided characteristics. For example, when the fingerprint is tainted/unclean or the facial features (in face recognition scenarios) diverge from the original data. Consequently, maintaining a certain degree of ”flexibility” or leniency within the system becomes crucial, especially when aiming for a specific *accuracy* level. The accuracy of a model is calculated as follows

$$A = \frac{Matched}{N} \times 100\%$$

where:

- `A`: is the accuracy
- `matched`: Number of correct matched items (e.g., Pixels on a fingerprint)
- `N`: Total Number of items in the set (e.g., total number of Pixels on a fingerprint)

To illustrate this concept within the context of Part III, we will embrace an approximated model that evaluates the accuracy of the utilized fingerprint. The permissible error threshold ($\varepsilon$) will signify the percentage of **unmatched** pixels in the fingerprint. This symbol $\varepsilon$ serves as an indicator of the degree to which discrepancies can be accommodated before a ”failure” is recorded.

For instance, consider a scenario where the initial fingerprint is depicted by a 10 x 10 pixel matrix (even though quite small) as shown in Figure 6.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | × | ✓ | ✓ |
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | × | × | ✓ | ✓ |
| ✓ | ✓ | ✓ | ✓ | ✓ | × | ✓ | ✓ | ✓ | ✓ |
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | × | ✓ | ✓ |
| ✓ | ✓ | × | ✓ | ✓ | ✓ | ✓ | × | ✓ | ✓ |
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| ✓ | × | × | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| ✓ | ✓ | × | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Figure 6: Left matrix represents of a perfect matching pixels (100% accuracy and $\varepsilon = 0\%$) Fingerprint. Right matrix represents of a 90% accuracy match and $\varepsilon = 10\%$ Fingerprint.

A stringent system would demand a pixel-to-pixel conformity between the input and original matrices. It's important to recognize that achieving a flawless 100% accuracy under such circumstances is impossible. In this context, when $\varepsilon = 0$ it signifies a perfect match is expected as shown in the left matrix. Now, consider that $\varepsilon = 10\%$, then this implies a tolerance for *deviations*. In practical terms, it allows for a maximum of 90 matching pixels with the original as shown in the right matrix. This situation can be analogized to a scenario where, let's say, a person is handling a delicious greasy taco before providing their fingerprint into a biometric device; in this case, certain pixels might not align due to the grease.

Given these considerations, adjustments must be made to the model presented in Part II. Referring to Figure 7, the necessary modifications will encompass the following changes:

- *define* a global variable called `epsilon` that will represent the allowed imprecision allowing to fail during the scanning.
- *compute* the accuracy of each time a fingerprint is provided and compare it against the original fingerprint representation.
- *test* the program using different fingerprint objects.

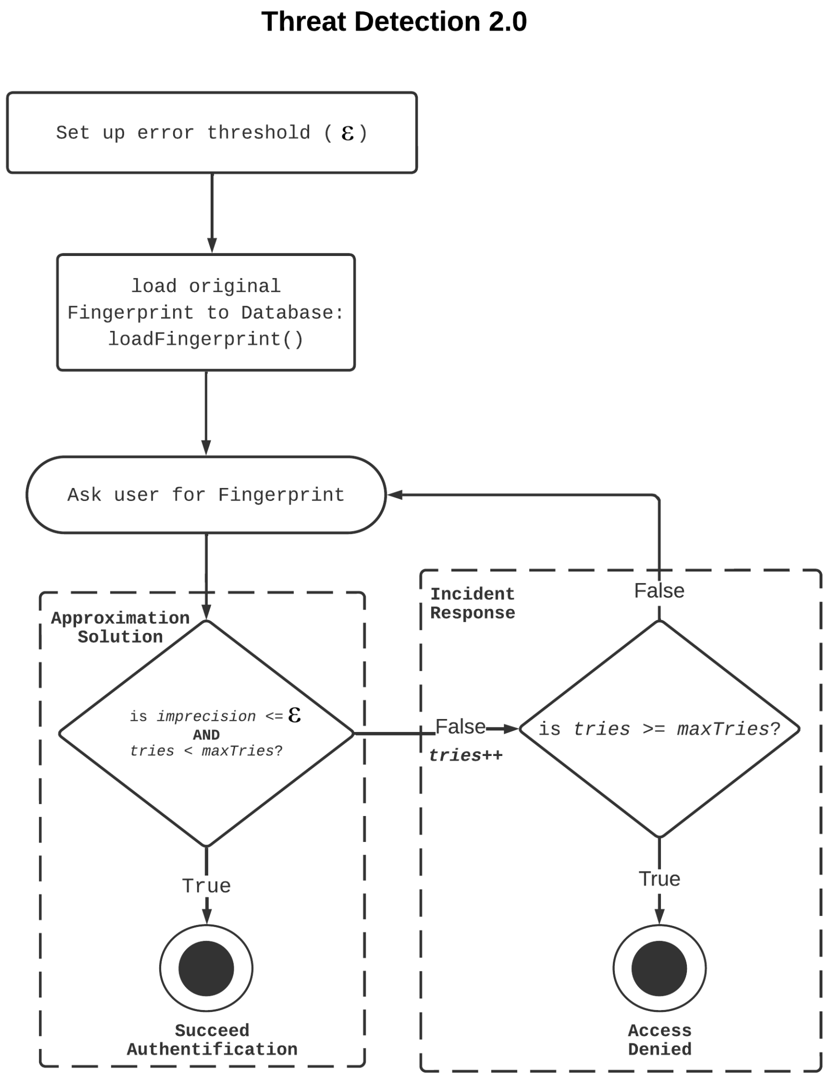


Figure 7: Model 3 – Applying Approximation and an Incident Response Model to Address Biometric Authentication Failure