جامعة جدة
University of Jeddah

Software Security Project Phase 1

Section: CY8

Made by:

Razan Alghanmi – 2111007

Shahad Kulaibi – 2114565

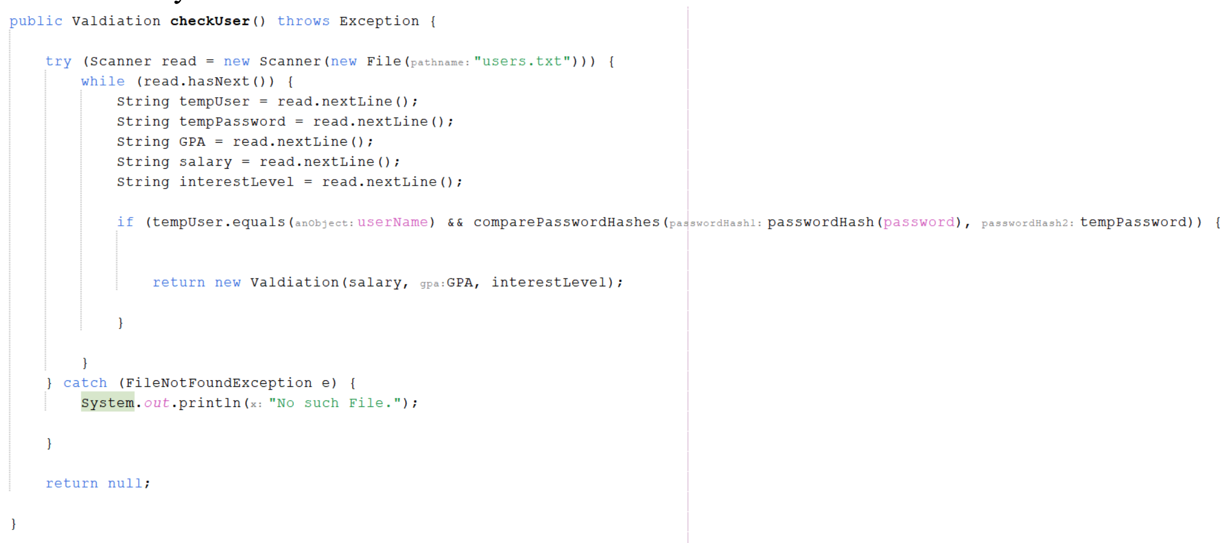
Elaf Bahashwan – 2110527

1.1 Security Design Principles:

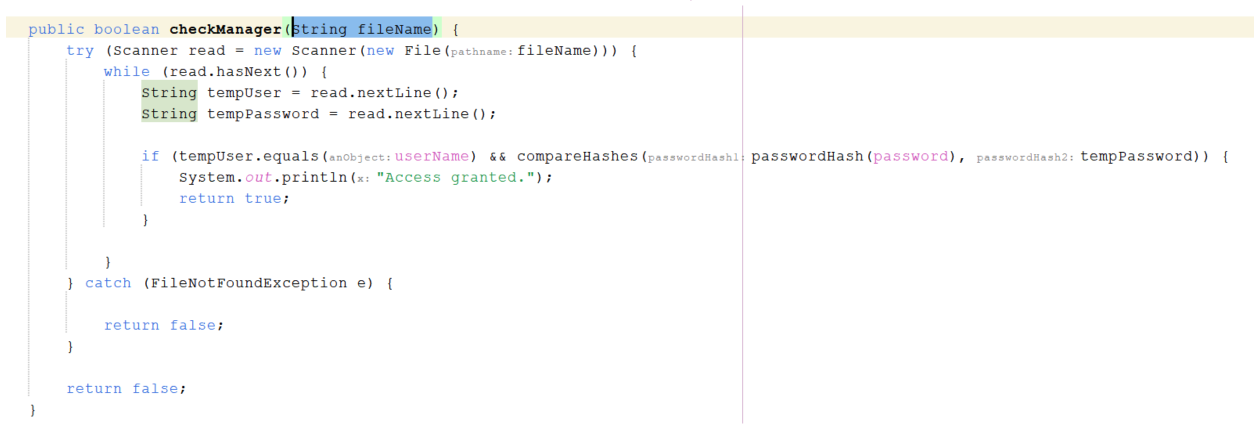
We implemented 8 security design principles, and the justification is as follows:

1- Access Control

The access control process typically involves user authentication, which authenticates the user's (either a user or manager) identity through credentials of usernames and passwords. Once authenticated, this can ensure that users only have access to the resources necessary for their designated tasks while ensuring the security and confidentiality of sensitive information.



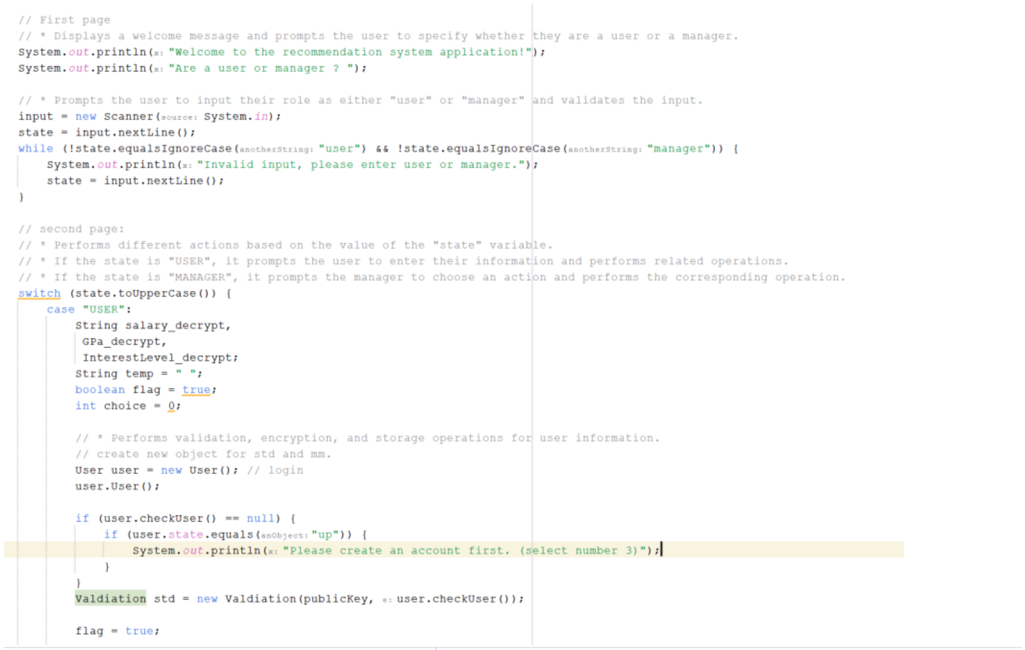
```
public Valdiation checkUser() throws Exception {

    try (Scanner read = new Scanner(new File(pathname: "users.txt"))) {
        while (read.hasNext()) {
            String tempUser = read.nextLine();
            String tempPassword = read.nextLine();
            String GPA = read.nextLine();
            String salary = read.nextLine();
            String interestLevel = read.nextLine();

            if (tempUser.equals(anObject: userName) && comparePasswordHashes(passwordHash1: passwordHash(password), passwordHash2: tempPassword)) {


                return new Valdiation(salary, gpa:GPA, interestLevel);

            }

        }
    } catch (FileNotFoundException e) {
        System.out.println(x: "No such File.");

    }

    return null;

}
```



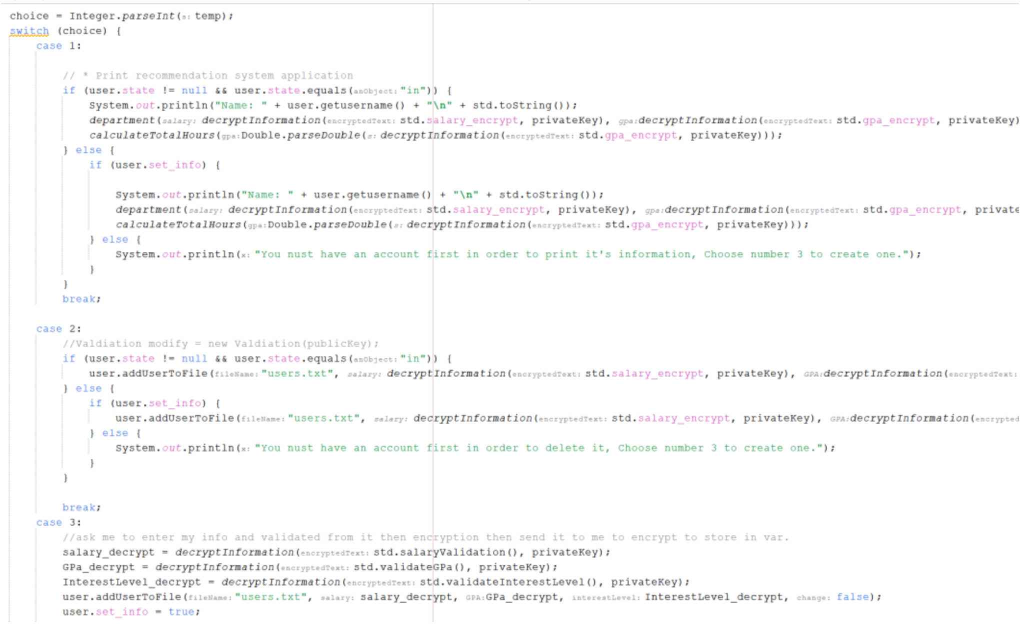
```
public boolean checkManager(String fileName) {
    try (Scanner read = new Scanner(new File(pathname: fileName))) {
        while (read.hasNext()) {
            String tempUser = read.nextLine();
            String tempPassword = read.nextLine();

            if (tempUser.equals(anObject: userName) && compareHashes(passwordHash1: passwordHash(password), passwordHash2: tempPassword)) {
                System.out.println(x: "Access granted.");
                return true;
            }

        }
    } catch (FileNotFoundException e) {

        return false;
    }

    return false;
}
```

2- Least Privilege

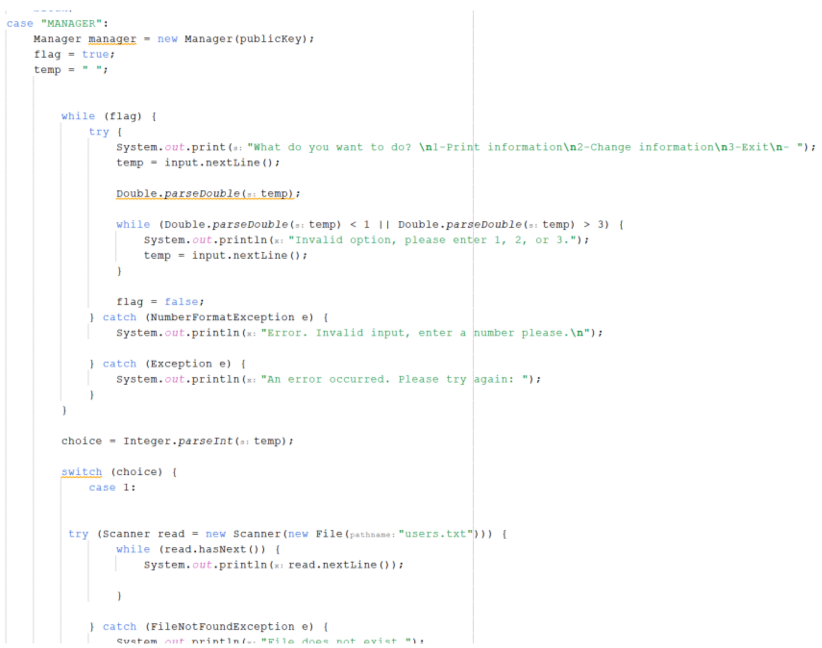
In this relationship, this principle ensures that users are only granted the minimum level of access required to perform their tasks, minimizing the potential risks associated with unauthorized access. For example, the manager can see the file that stores all of the system user's information. Also, they can change the requirements for every department. On the other hand, users can only see their information, and delete their account.



```java
// First page
// * Displays a welcome message and prompts the user to specify whether they are a user or a manager.
System.out.println(x: "Welcome to the recommendation system application!");
System.out.println(x: "Are a user or manager ? ");

// * Prompts the user to input their role as either "user" or "manager" and validates the input.
input = new Scanner(source: System.in);
state = input.nextLine();
while (!state.equalsIgnoreCase(anotherString: "user") && !state.equalsIgnoreCase(anotherString: "manager")) {
    System.out.println(x: "Invalid input, please enter user or manager.");
    state = input.nextLine();
}

// second page:
// * Performs different actions based on the value of the "state" variable.
// * If the state is "USER", it prompts the user to enter their information and performs related operations.
// * If the state is "MANAGER", it prompts the manager to choose an action and performs the corresponding operation.
switch (state.toUpperCase()) {
    case "USER":
        String salary_decrypt,
         GPa_decrypt,
         InterestLevel_decrypt;
        String temp = " ";
        boolean flag = true;
        int choice = 0;

        // * Performs validation, encryption, and storage operations for user information.
        // create new object for std and mm.
        User user = new User(); // login
        user.User();

        if (user.checkUser() == null) {
            if (user.state.equals(anObject:"up")) {
                System.out.println(x: "Please create an account first. (select number 3)");
            }
        }
        Valdiation std = new Valdiation(publicKey, e: user.checkUser());

        flag = true;
```



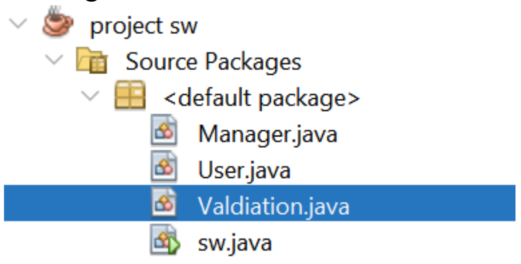
```java
choice = Integer.parseInt(s: temp);
switch (choice) {
    case 1:

        // * Print recommendation system application
        if (user.state != null && user.state.equals(anObject:"in")) {
            System.out.println("Name: " + user.getusername() + "\n" + std.toString());
            department(salary: decryptInformation(encryptedText: std.salary_encrypt, privateKey), gpa:decryptInformation(encryptedText: std.gpa_encrypt, privateKey)
            calculateTotalHours(gpa:Double.parseDouble(s: decryptInformation(encryptedText: std.gpa_encrypt, privateKey)));
        } else {
            if (user.set_info) {

                System.out.println("Name: " + user.getusername() + "\n" + std.toString());
                department(salary: decryptInformation(encryptedText: std.salary_encrypt, privateKey), gpa:decryptInformation(encryptedText: std.gpa_encrypt, private
                calculateTotalHours(gpa:Double.parseDouble(s: decryptInformation(encryptedText: std.gpa_encrypt, privateKey)));
            } else {
                System.out.println(x: "You nust have an account first in order to print it's information, Choose number 3 to create one.");
            }
        }
        break;

    case 2:
        //Valdiation modify = new Valdiation(publicKey);
        if (user.state != null && user.state.equals(anObject:"in")) {
            user.addUserToFile(fileName:"users.txt", salary: decryptInformation(encryptedText: std.salary_encrypt, privateKey), GPA:decryptInformation(encryptedText:
        } else {
            if (user.set_info) {
                user.addUserToFile(fileName:"users.txt", salary: decryptInformation(encryptedText: std.salary_encrypt, privateKey), GPA:decryptInformation(encrypted
            } else {
                System.out.println(x: "You nust have an account first in order to delete it, Choose number 3 to create one.");
            }
        }

        break;
    case 3:
        //ask me to enter my info and validated from it then encryption then send it to me to encrypt to store in var.
        salary_decrypt = decryptInformation(encryptedText: std.salaryValidation(), privateKey);
        GPa_decrypt = decryptInformation(encryptedText: std.validateGPa(), privateKey);
        InterestLevel_decrypt = decryptInformation(encryptedText: std.validateInterestLevel(), privateKey);
        user.addUserToFile(fileName:"users.txt", salary: salary_decrypt, GPA:GPa_decrypt, interestLevel: InterestLevel_decrypt, change: false);
        user.set_info = true;
```



```
case "MANAGER":
    Manager manager = new Manager(publicKey);
    flag = true;
    temp = " ";


        while (flag) {
            try {
                System.out.print(s: "What do you want to do? \n1-Print information\n2-Change information\n3-Exit\n- ");
                temp = input.nextLine();

                Double.parseDouble(s: temp);

                while (Double.parseDouble(s: temp) < 1 || Double.parseDouble(s: temp) > 3) {
                    System.out.println(x: "Invalid option, please enter 1, 2, or 3.");
                    temp = input.nextLine();
                }

                flag = false;
            } catch (NumberFormatException e) {
                System.out.println(x: "Error. Invalid input, enter a number please.\n");

            } catch (Exception e) {
                System.out.println(x: "An error occurred. Please try again: ");
            }
        }

        choice = Integer.parseInt(s: temp);

        switch (choice) {
            case 1:


         try (Scanner read = new Scanner(new File(pathname: "users.txt"))) {
                while (read.hasNext()) {
                    System.out.println(x: read.nextLine());

                }

            } catch (FileNotFoundException e) {
                System.out.println(x: "File does not exist ");
```

3- Compartmentalization

Compartmentalization was achieved by using separate classes for handling user and manager functionalities, as well as for validation.
The user and manager functionalities can be implemented in separate classes, each responsible for performing operations specific to their respective roles. For example, the user class can handle operations related to user-specific tasks such as accessing files, managing personal information, or performing actions based on user privileges. Similarly, the manager class can handle tasks specific to managers. By compartmentalizing the code into separate classes, it becomes easier to manage and maintain different functionalities. Each class can focus on its specific responsibilities, making the code more modular and reusable.



```
project sw
    Source Packages
        <default package>
            Manager.java
            User.java
            Valdiation.java
            sw.java
```

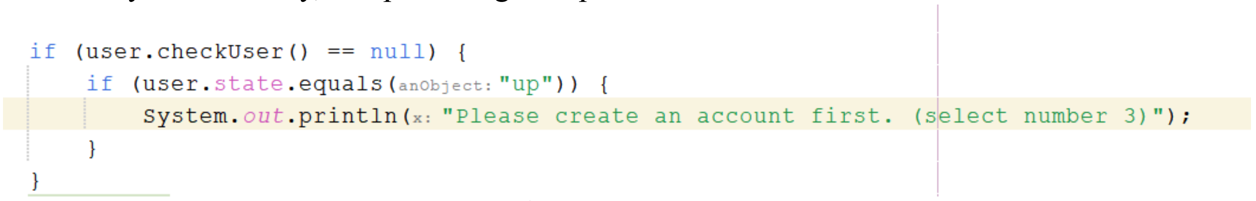
4- Fail Safe Default

In the context of user and manager functionalities, the use of fail-safe defaults can provide an added layer of security and prevent unintended actions or access. Fail-safe defaults refer to setting default values or configurations that prioritize safety and minimize potential risks.

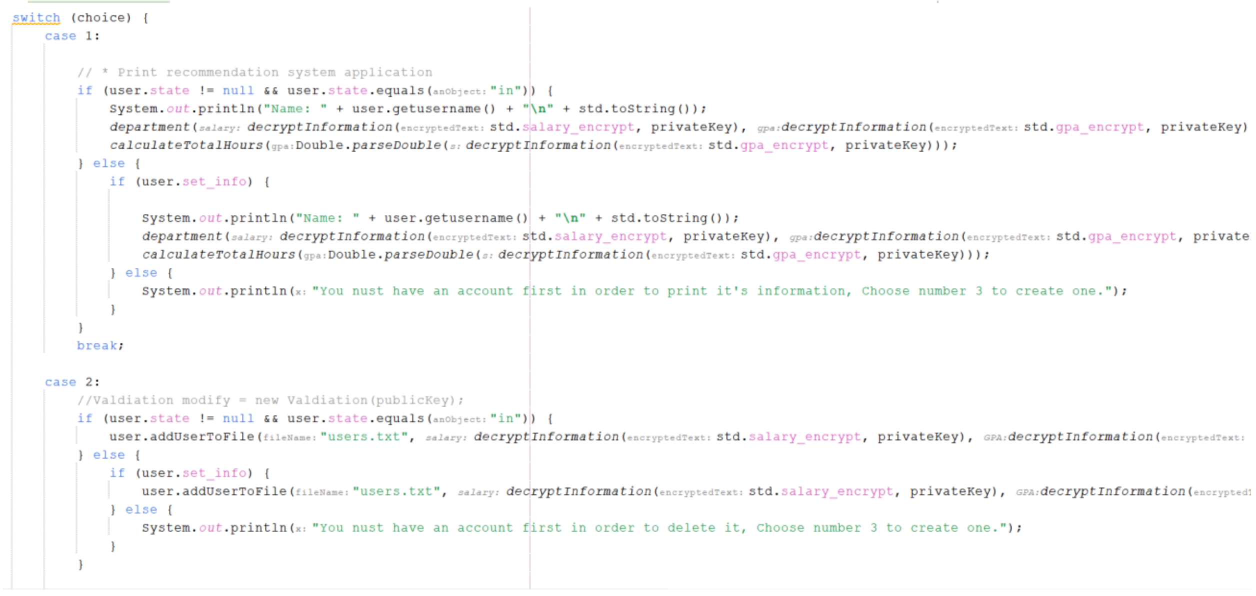
For example, in the user class, fail-safe defaults can be implemented to restrict certain actions or limit access to sensitive information by default. This ensures that users cannot accidentally or maliciously perform actions that could compromise the system's integrity or violate privacy.

Similarly, in the manager class, fail-safe defaults can be employed to prevent unauthorized changes or accidental granting of excessive permissions. By setting default configurations that prioritize caution and restrict certain actions, the risk of unintended consequences or security breaches can be significantly reduced.

In both cases, fail-safe defaults act as a safeguard against potential errors or misuse. They provide a baseline level of security and control, ensuring that users and managers operate within predefined boundaries. This approach helps to mitigate risks, enhance system stability, and protect against potential vulnerabilities.



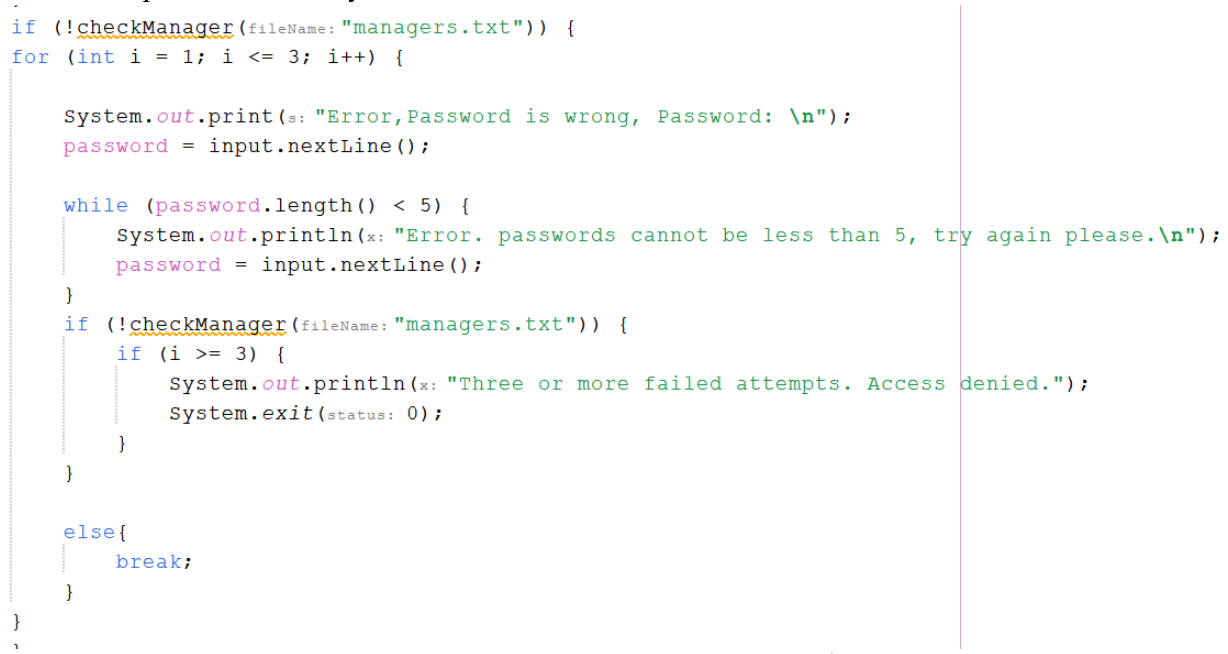
```
if (user.checkUser() == null) {
    if (user.state.equals(anObject: "up")) {
        System.out.println(x: "Please create an account first. (select number 3)");
    }
}
```



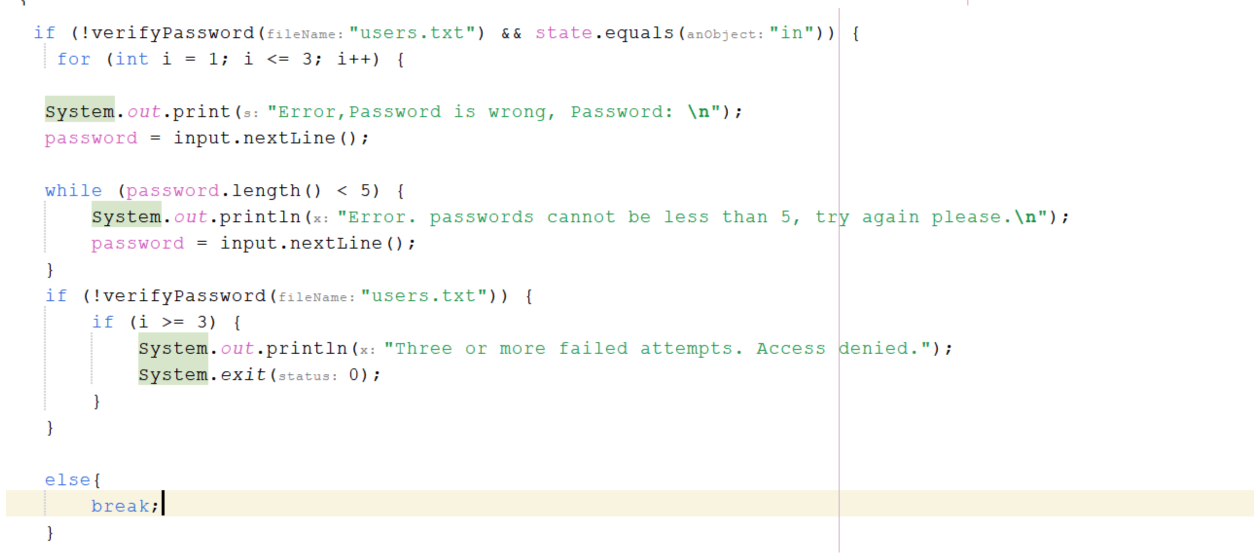
```
switch (choice) {
    case 1:

        // * Print recommendation system application
        if (user.state != null && user.state.equals(anObject: "in")) {
            System.out.println("Name: " + user.getusername() + "\n" + std.toString());
            department(salary: decryptInformation(encryptedText: std.salary_encrypt, privateKey), gpa:decryptInformation(encryptedText: std.gpa_encrypt, privateKey)
            calculateTotalHours(gpa:Double.parseDouble(s: decryptInformation(encryptedText: std.gpa_encrypt, privateKey)));
        } else {
            if (user.set_info) {

                System.out.println("Name: " + user.getusername() + "\n" + std.toString());
                department(salary: decryptInformation(encryptedText: std.salary_encrypt, privateKey), gpa:decryptInformation(encryptedText: std.gpa_encrypt, private
                calculateTotalHours(gpa:Double.parseDouble(s: decryptInformation(encryptedText: std.gpa_encrypt, privateKey)));
            } else {
                System.out.println(x: "You nust have an account first in order to print it's information, Choose number 3 to create one.");
            }
        }
        break;

    case 2:
        //Valdiation modify = new Valdiation(publicKey);
        if (user.state != null && user.state.equals(anObject: "in")) {
            user.addUserToFile(fileName: "users.txt", salary: decryptInformation(encryptedText: std.salary_encrypt, privateKey), GPA:decryptInformation(encryptedText:
        } else {
            if (user.set_info) {
                user.addUserToFile(fileName: "users.txt", salary: decryptInformation(encryptedText: std.salary_encrypt, privateKey), GPA:decryptInformation(encryptedT
            } else {
                System.out.println(x: "You nust have an account first in order to delete it, Choose number 3 to create one.");
            }
        }
```

5- Fail Secure

The use of fail secure mechanism can provide an extra layer of security by implementing a strict limit on password attempts. For example, if a user or manager fails to enter the correct password after three attempts, the system will automatically exit. This fail-secure approach prevents unauthorized access attempts by limiting the number of chances an individual has to guess or crack a password. By implementing such a mechanism, the system ensures that only authorized users or managers with the correct credentials can gain access, thereby reducing the risk of unauthorized access and potential security breaches.



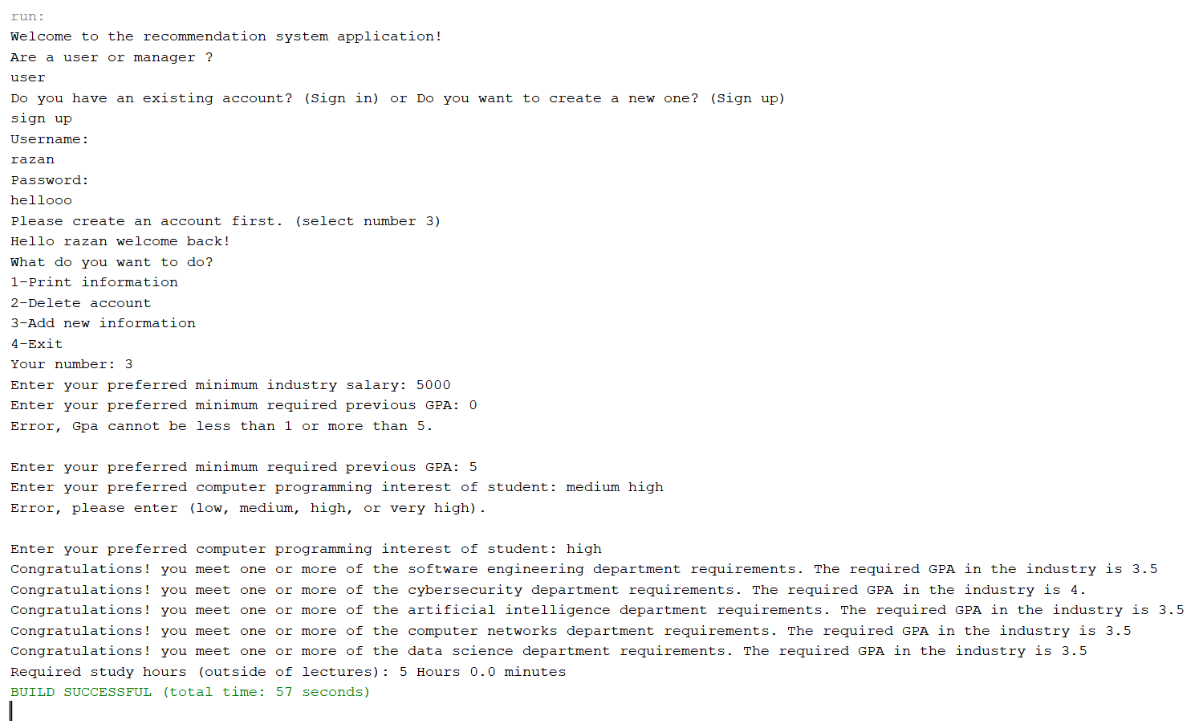
```java
if (!checkManager(fileName: "managers.txt")) {
for (int i = 1; i <= 3; i++) {

    System.out.print(s: "Error,Password is wrong, Password: \n");
    password = input.nextLine();

    while (password.length() < 5) {
        System.out.println(x: "Error. passwords cannot be less than 5, try again please.\n");
        password = input.nextLine();
    }
    if (!checkManager(fileName: "managers.txt")) {
        if (i >= 3) {
            System.out.println(x: "Three or more failed attempts. Access denied.");
            System.exit(status: 0);
        }
    }

    else{
        break;
    }
}
```



```java
 if (!verifyPassword(fileName: "users.txt") && state.equals(anObject: "in")) {
   for (int i = 1; i <= 3; i++) {

  System.out.print(s: "Error,Password is wrong, Password: \n");
  password = input.nextLine();

  while (password.length() < 5) {
      System.out.println(x: "Error. passwords cannot be less than 5, try again please.\n");
      password = input.nextLine();
  }
  if (!verifyPassword(fileName: "users.txt")) {
      if (i >= 3) {
          System.out.println(x: "Three or more failed attempts. Access denied.");
          System.exit(status: 0);
      }
  }

  else{
      break;
  }
```

6- Least Astonishment

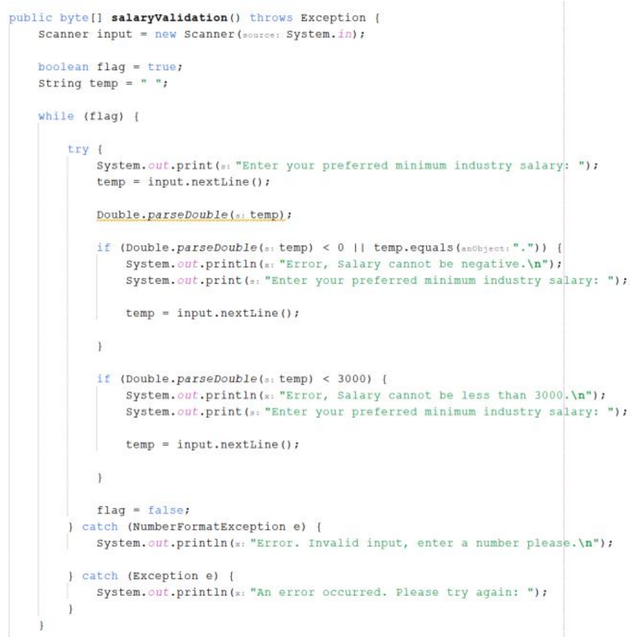
The principle of Least Astonishment plays a crucial role in ensuring a smooth and intuitive user experience. This principle suggests that a system should behave in a way that is least surprising or unexpected to its users. It is important to organize and present information in a manner that aligns with their mental models and expectations. Also, if errors arise, a clear description of the error and why it happened is displayed. By doing so, they can easily navigate through the system, locate desired features or information, and perform tasks efficiently. Additionally, adhering to the principle of Least Astonishment helps minimize confusion and frustration, ultimately enhancing user satisfaction and productivity.



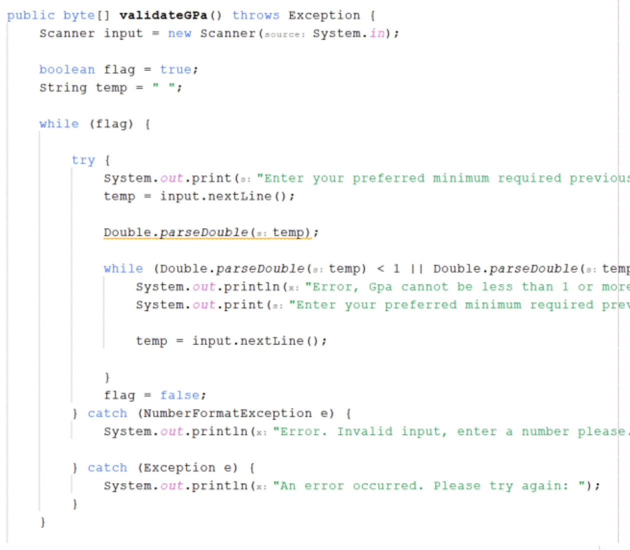
```
run:
Welcome to the recommendation system application!
Are a user or manager ?
user
Do you have an existing account? (Sign in) or Do you want to create a new one? (Sign up)
sign up
Username:
razan
Password:
hellooo
Please create an account first. (select number 3)
Hello razan welcome back!
What do you want to do?
1-Print information
2-Delete account
3-Add new information
4-Exit
Your number: 3
Enter your preferred minimum industry salary: 5000
Enter your preferred minimum required previous GPA: 0
Error, Gpa cannot be less than 1 or more than 5.

Enter your preferred minimum required previous GPA: 5
Enter your preferred computer programming interest of student: medium high
Error, please enter (low, medium, high, or very high).

Enter your preferred computer programming interest of student: high
Congratulations! you meet one or more of the software engineering department requirements. The required GPA in the industry is 3.5
Congratulations! you meet one or more of the cybersecurity department requirements. The required GPA in the industry is 4.
Congratulations! you meet one or more of the artificial intelligence department requirements. The required GPA in the industry is 3.5
Congratulations! you meet one or more of the computer networks department requirements. The required GPA in the industry is 3.5
Congratulations! you meet one or more of the data science department requirements. The required GPA in the industry is 3.5
Required study hours (outside of lectures): 5 Hours 0.0 minutes
BUILD SUCCESSFUL (total time: 57 seconds)
```

7- Minimize Trust Surface

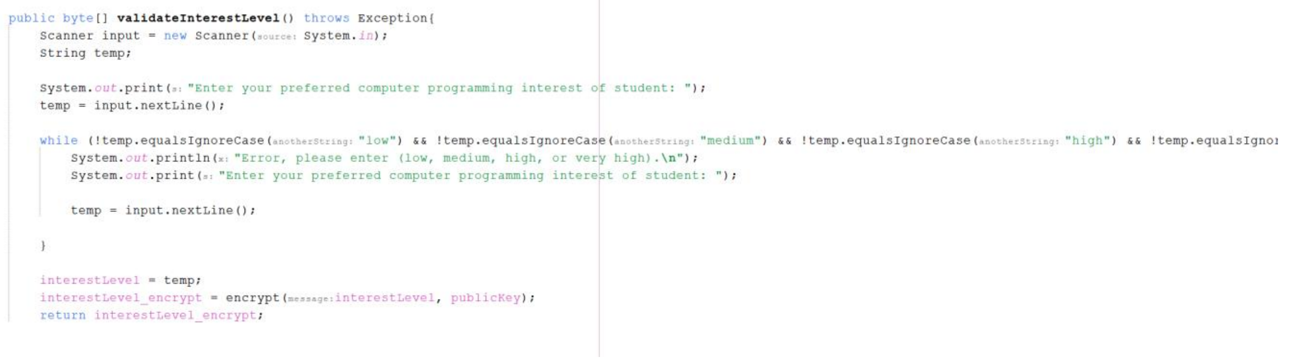
Minimizing the trust surface is crucial for maintaining security and protecting sensitive information. The concept of minimizing the trust surface involves reducing the number of potential attack vectors or points of vulnerability within a system. This can be achieved by implementing strong authentication and authorization mechanisms, limiting access privileges to only necessary functions and data. This was achieved by excessive input validation.



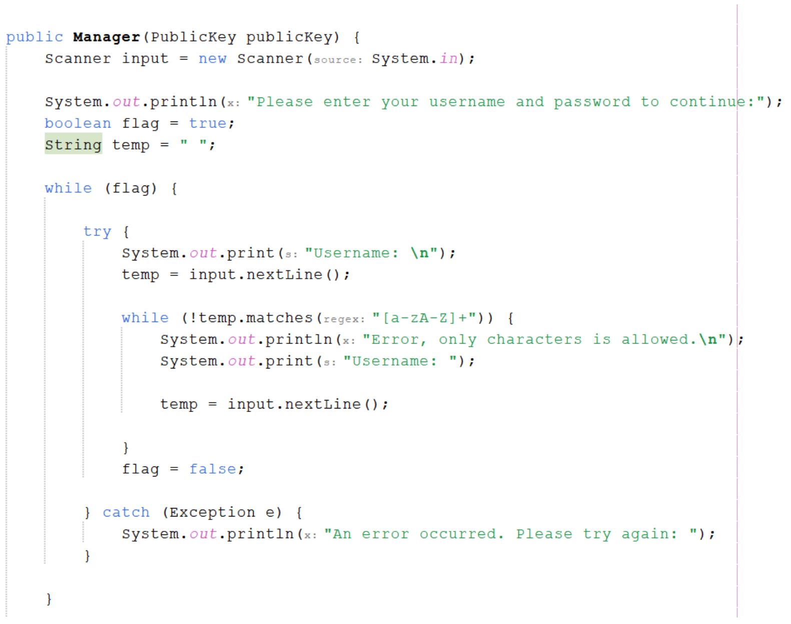
```java
public byte[] salaryValidation() throws Exception {
    Scanner input = new Scanner(source: System.in);

    boolean flag = true;
    String temp = " ";

    while (flag) {

        try {
            System.out.print(s: "Enter your preferred minimum industry salary: ");
            temp = input.nextLine();

            Double.parseDouble(s: temp);

            if (Double.parseDouble(s: temp) < 0 || temp.equals(anObject: ".")) {
                System.out.println(x: "Error, Salary cannot be negative.\n");
                System.out.print(s: "Enter your preferred minimum industry salary: ");

                temp = input.nextLine();

            }

            if (Double.parseDouble(s: temp) < 3000) {
                System.out.println(x: "Error, Salary cannot be less than 3000.\n");
                System.out.print(s: "Enter your preferred minimum industry salary: ");

                temp = input.nextLine();

            }

            flag = false;
        } catch (NumberFormatException e) {
            System.out.println(x: "Error. Invalid input, enter a number please.\n");

        } catch (Exception e) {
            System.out.println(x: "An error occurred. Please try again: ");
        }
    }
```



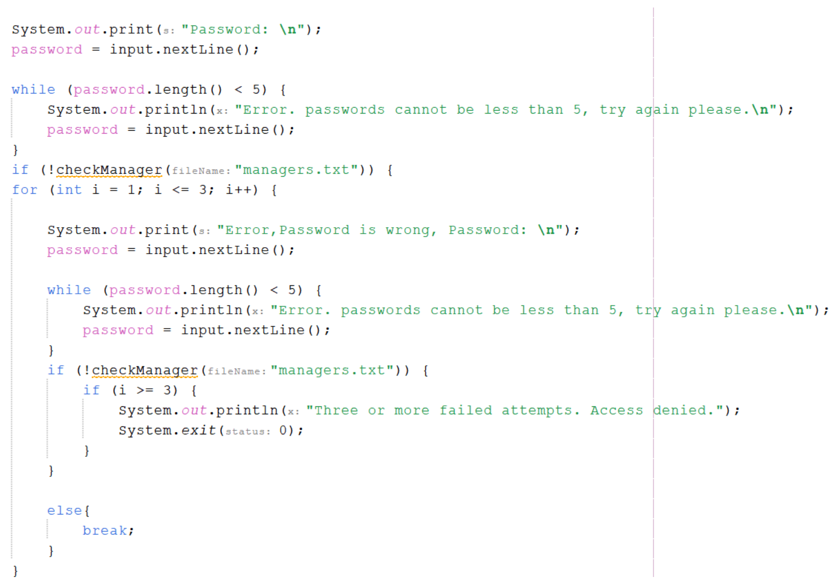
```java
public byte[] validateGPa() throws Exception {
    Scanner input = new Scanner(source: System.in);

    boolean flag = true;
    String temp = " ";

    while (flag) {

        try {
            System.out.print(s: "Enter your preferred minimum required previous GPA: ");
            temp = input.nextLine();

            Double.parseDouble(s: temp);

            while (Double.parseDouble(s: temp) < 1 || Double.parseDouble(s: temp) > 5) {
                System.out.println(x: "Error, Gpa cannot be less than 1 or more than 5.\n");
                System.out.print(s: "Enter your preferred minimum required previous GPA: ");

                temp = input.nextLine();

            }
            flag = false;
        } catch (NumberFormatException e) {
            System.out.println(x: "Error. Invalid input, enter a number please.\n");

        } catch (Exception e) {
            System.out.println(x: "An error occurred. Please try again: ");
        }
    }
```



```java
public byte[] validateInterestLevel() throws Exception{
    Scanner input = new Scanner(source: System.in);
    String temp;

    System.out.print(s: "Enter your preferred computer programming interest of student: ");
    temp = input.nextLine();

    while (!temp.equalsIgnoreCase(anotherString: "low") && !temp.equalsIgnoreCase(anotherString: "medium") && !temp.equalsIgnoreCase(anotherString: "high") && !temp.equalsIgnoı
        System.out.println(x: "Error, please enter (low, medium, high, or very high).\n");
        System.out.print(s: "Enter your preferred computer programming interest of student: ");

        temp = input.nextLine();

    }

    interestLevel = temp;
    interestLevel_encrypt = encrypt(message:interestLevel, publicKey);
    return interestLevel_encrypt;
```



```java
public Manager(PublicKey publicKey) {
    Scanner input = new Scanner(source: System.in);

    System.out.println(x: "Please enter your username and password to continue:");
    boolean flag = true;
    String temp = " ";

    while (flag) {

        try {
            System.out.print(s: "Username: \n");
            temp = input.nextLine();

            while (!temp.matches(regex: "[a-zA-Z]+")) {
                System.out.println(x: "Error, only characters is allowed.\n");
                System.out.print(s: "Username: ");

                temp = input.nextLine();

            }
            flag = false;

        } catch (Exception e) {
            System.out.println(x: "An error occurred. Please try again: ");
        }

    }
```

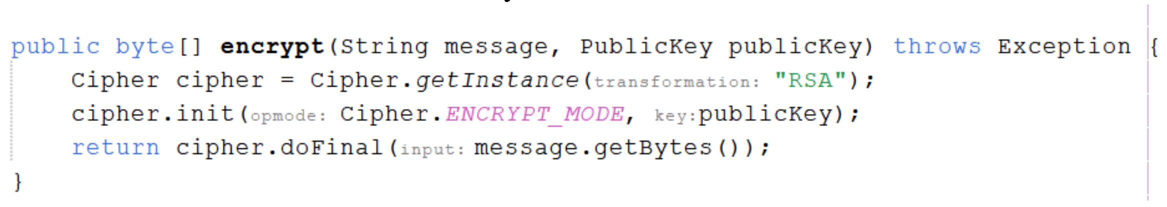


```
System.out.print(s: "Password: \n");
password = input.nextLine();

while (password.length() < 5) {
    System.out.println(x: "Error. passwords cannot be less than 5, try again please.\n");
    password = input.nextLine();
}
if (!checkManager(fileName: "managers.txt")) {
for (int i = 1; i <= 3; i++) {

    System.out.print(s: "Error,Password is wrong, Password: \n");
    password = input.nextLine();

    while (password.length() < 5) {
        System.out.println(x: "Error. passwords cannot be less than 5, try again please.\n");
        password = input.nextLine();
    }
    if (!checkManager(fileName: "managers.txt")) {
        if (i >= 3) {
            System.out.println(x: "Three or more failed attempts. Access denied.");
            System.exit(status: 0);
        }
    }

    else{
        break;
    }
}
```

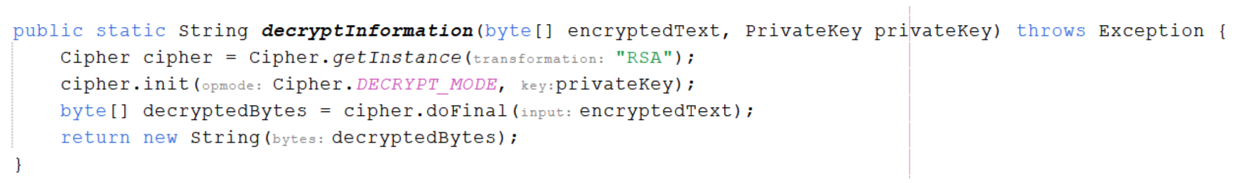
1.2 Security Requirements:

1- Confidentiality

Encryption plays a crucial role in maintaining confidentiality. By employing encryption techniques, sensitive information can be converted into an unreadable form that can only be deciphered with the appropriate decryption key. When data is returned from methods of the classes to the main method, they are encrypted. This ensures that even if unauthorized individuals gain access to the data, they will be unable to comprehend or exploit it. The main method then decrypts the data for its use. Encryption adds an extra layer of protection. It guards against eavesdropping and unauthorized access, guaranteeing that the information remains confidential and secure. The utilization of encryption not only enhances the reliability of the system but also provides reassurance to users and managers that their sensitive data is being handled with the utmost care and security.



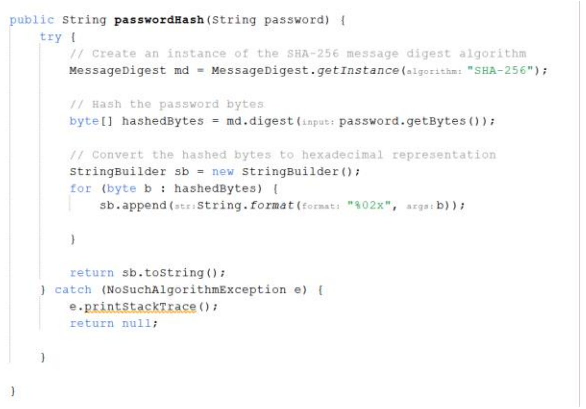
```
public byte[] encrypt(String message, PublicKey publicKey) throws Exception {
    Cipher cipher = Cipher.getInstance(transformation: "RSA");
    cipher.init(opmode: Cipher.ENCRYPT_MODE, key:publicKey);
    return cipher.doFinal(input: message.getBytes());
}
```



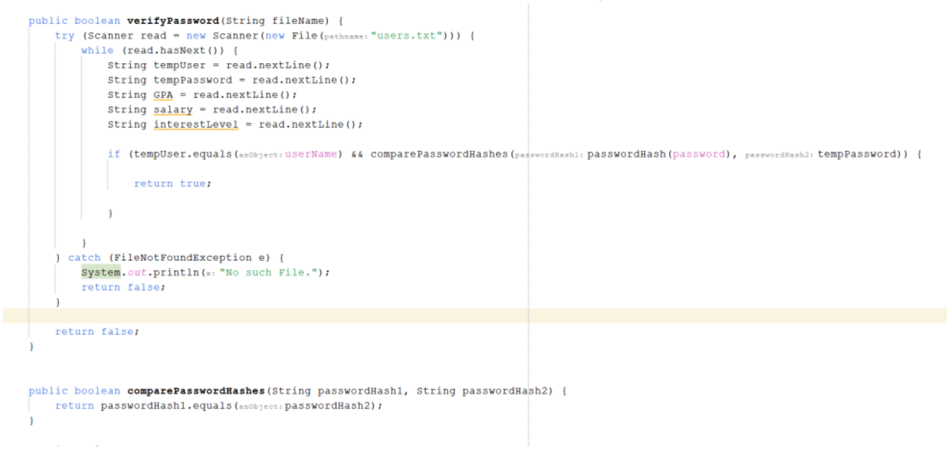
```
public static String decryptInformation(byte[] encryptedText, PrivateKey privateKey) throws Exception {
    Cipher cipher = Cipher.getInstance(transformation: "RSA");
    cipher.init(opmode: Cipher.DECRYPT_MODE, key:privateKey);
    byte[] decryptedBytes = cipher.doFinal(input: encryptedText);
    return new String(bytes: decryptedBytes);
}
```

2- Integrity

Using hashing for passwords is crucial in maintaining the integrity of data. Hashing is a one-way process that converts a password into a fixed-length string of characters called a hash. This ensures that the original password cannot be determined from the hash, adding an extra layer of security. When a user inputs their password, it is hashed and compared to the stored hash value. If the two hashes match, access is granted. The utilization of hashing for passwords helps prevent unauthorized entry and safeguards against potential attacks like brute-force or dictionary attacks, where attackers systematically try different combinations to guess passwords. By employing hashing, passwords are securely stored and authenticated without exposing the actual password, guaranteeing the reliability of user credentials and upholding the overall system's security.



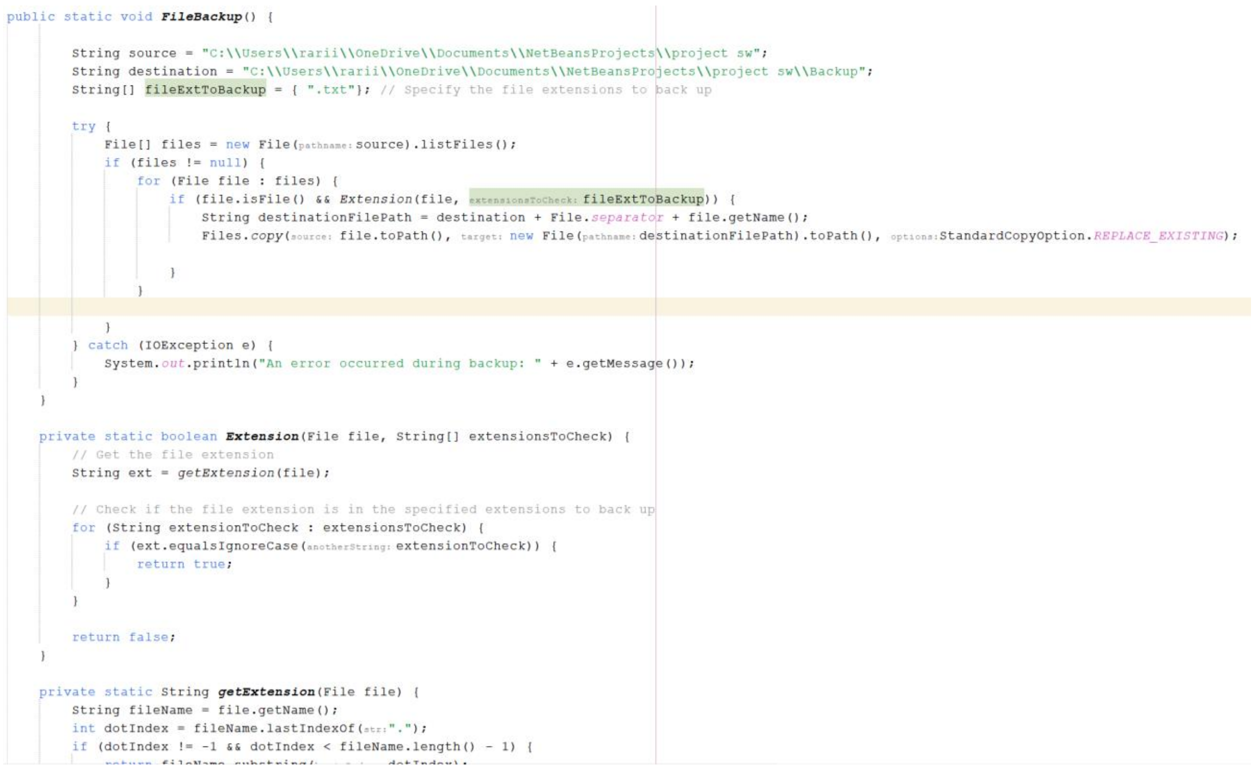
```java
public String passwordHash(String password) {
    try {
        // Create an instance of the SHA-256 message digest algorithm
        MessageDigest md = MessageDigest.getInstance(algorithm: "SHA-256");

        // Hash the password bytes
        byte[] hashedBytes = md.digest(input: password.getBytes());

        // Convert the hashed bytes to hexadecimal representation
        StringBuilder sb = new StringBuilder();
        for (byte b : hashedBytes) {
            sb.append(str:String.format(format: "%02x", args:b));

        }

        return sb.toString();
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
        return null;

    }

}
```



```java
public boolean verifyPassword(String fileName) {
    try (Scanner read = new Scanner(new File(pathname: "users.txt"))) {
        while (read.hasNext()) {
            String tempUser = read.nextLine();
            String tempPassword = read.nextLine();
            String GPA = read.nextLine();
            String salary = read.nextLine();
            String interestLevel = read.nextLine();

            if (tempUser.equals(anObject: userName) && comparePasswordHashes(passwordHash1: passwordHash(password), passwordHash2: tempPassword)) {

                return true;

            }

        }
    } catch (FileNotFoundException e) {
        System.out.println(x: "No such File.");
        return false;
    }

    return false;
}


public boolean comparePasswordHashes(String passwordHash1, String passwordHash2) {
    return passwordHash1.equals(anObject: passwordHash2);
}
```

3- Availability

The utilization of file backups is crucial to ensure data availability. Backing up files involves creating duplicates of the user and manager files. This practice guarantees that if the original files are lost or damaged, they can be easily restored from the backup copies. Regularly backing up files helps organizations minimize the risk of data loss and ensures that vital information remains accessible. In the event of hardware failures, natural disasters, or cyber-attacks, having backups enables swift recovery and reduces downtime. It also provides protection against accidental deletions or human errors. By implementing a strong backup strategy, we can ensure file availability, maintain business continuity, and protect against potential disruptions or data loss incidents.



```java
public static void FileBackup() {

        String source = "C:\\Users\\rarii\\OneDrive\\Documents\\NetBeansProjects\\project sw";
        String destination = "C:\\Users\\rarii\\OneDrive\\Documents\\NetBeansProjects\\project sw\\Backup";
        String[] fileExtToBackup = { ".txt"}; // Specify the file extensions to back up

        try {
            File[] files = new File(pathname: source).listFiles();
            if (files != null) {
                for (File file : files) {
                    if (file.isFile() && Extension(file, extensionsToCheck: fileExtToBackup)) {
                        String destinationFilePath = destination + File.separator + file.getName();
                        Files.copy(source: file.toPath(), target: new File(pathname: destinationFilePath).toPath(), options:StandardCopyOption.REPLACE_EXISTING);

                    }
                }

            }
        } catch (IOException e) {
            System.out.println("An error occurred during backup: " + e.getMessage());
        }
    }

    private static boolean Extension(File file, String[] extensionsToCheck) {
        // Get the file extension
        String ext = getExtension(file);

        // Check if the file extension is in the specified extensions to back up
        for (String extensionToCheck : extensionsToCheck) {
            if (ext.equalsIgnoreCase(anotherString: extensionToCheck)) {
                return true;
            }
        }

        return false;
    }

    private static String getExtension(File file) {
        String fileName = file.getName();
        int dotIndex = fileName.lastIndexOf(str:".");
        if (dotIndex != -1 && dotIndex < fileName.length() - 1) {
```