```java
package use_case_controller;

import java.util.HashMap;

/**
 * Class LogInController to manage various validation and control flows of use case "Log In".
 * @author Daria Vekic (Student ID: 586661)
 *
 */
public class LogInController {

    //Instance fields
    private FileIO fileHandler = new FileIO(); //for file handling
    private CommonElements common = new CommonElements(); //for common View elements
    //Error messages
    public String errorUsername = "Username does not exist.";
    public String errorNoMatch = "Password does not match.";

    /**
     * Method to update a password value in the Map.
     * @param map the Map to be updated.
     * @param username the Key of the Value to be updated.
     * @param newPw the new password to be hashed.
     */
    private void changePassword(HashMap<String, String> map, String username, String newPw) {
        String salt = BCrypt.gensalt(10);
        String hash = BCrypt.hashpw(newPw, salt);
        map.replace(username, hash); //update the map with hash of new password
        fileHandler.writeToFile(map);
    } //end method changePassword

    /**
     * Method to check if password input by user is the same as the password in the Map.
     * @param map the HashMap containing login credentials.
     * @param username the Key in the Map.
     * @param input the password input by the user.
     * @return true if input matches hash value in the Map; false otherwise.
     */
    private boolean checkMatch(HashMap<String, String> map, String username, String input) {
        return BCrypt.checkpw(input, map.get(username)); //check input against hash in file
    } //end method checkMatch

    /**
     * Method to check if username input by user is contained in the Map.
     * @param input the username input by the user.
     * @param map the Map to check input against.
     * @return true if username is found in the Map; false otherwise.
     */
    private boolean checkUsername(String input, HashMap<String, String> map) {
        return map.containsKey(input); //returns true if username contained in the Map
    } //end method checkUsernameInMap

    /**
     * Method to control routine of resetting a user's password.
     * Checks a username exists. If it does, checks user's old password is current Value.
     * Validates the new password meets criteria according business rules.
     * If it does, the new password is hashed and stored in file.
     * @param usernameTxtField used to receive username input.
     * @param oldPwTxtField used to receive old password input.
     * @param newPwTxtField used to receive new password input.
     * @return true if password is successfully changed; false otherwise.
     */
    public boolean resetPassword(JTextField usernameTxtField, JPasswordField oldPwTxtField, JPasswordField newPwTxtField) {
        String username = usernameTxtField.getText();
        HashMap<String, String> mapFromFile = fileHandler.readFromFile();
        boolean exists = checkUsername(username, mapFromFile); //check username exists
        if(!exists) //if username not found
            common.showError(errorUsername, "Error");
        else { //if username exists
            String oldPw = String.valueOf(oldPwTxtField.getPassword()); //get old password
            boolean match = checkMatch(mapFromFile, username, oldPw); //check old password matches
            if(!match) { //if old password doesn't match
                common.showError("Old" + errorNoMatch.toLowerCase(), "Error");
            } else { //if old password is a match
                String newPw = String.valueOf(newPwTxtField.getPassword()); //get the new password
                boolean meetsCriteria = validateNewPassword(newPw); //check new password meets criteria in business rules
                if(!meetsCriteria) { //if new password doesn't meet criteria
                    common.showError("New password does not meet criteria.\nMust be minimum 12 characters"
                            + " and include at least 1 special character, 1 number, and 1 uppercase letter.", "Error");
                } else { //if new password is good
                    changePassword(mapFromFile, username, newPw); //update the Map
                    JOptionPane.showMessageDialog(null, "Password successfully changed.\nPress OK to return to Log In.",
                            "Success", JOptionPane.INFORMATION_MESSAGE);
                    return true;
                } //end if else
            } //end if else
        } //end if else
        return false;
    } //end method resetPassword

    /**
     * Method to control routine of signing in to system.
     * @param usernameTxtField used to receive username input.
     * @param pwField used to receive password input.
     * @return true if login details are valid; false otherwise.
     */
    public boolean signIn(JTextField usernameTxtField, JPasswordField pwField) {
        HashMap<String, String> map = fileHandler.readFromFile();
        String username = usernameTxtField.getText(); //get the username
        if(!checkUsername(username, map)) {
            common.showError(errorUsername, "Error");
        } else {
            String password = String.valueOf(pwField.getPassword());
            if(!checkMatch(map, username, password)) {
                common.showError(errorNoMatch, "Error");
            } else {
                return true;
            } //end if else
        } //end if else
        return false;
    } //end method signIn

    /**
     * Method to check new password meets criteria according to business rules.
     * Checks new password is minimum 12 characters and contains at least 1 special
     * character, 1 digit, and 1 uppercase letter.
     * @param newPassword the user's new password to validate.
     * @return true if new password is acceptable; false otherwise.
     */
    private boolean validateNewPassword(String newPassword) {
        boolean meetsCriteria = false;
        //check length
        if(newPassword.length() >= 12) {
            //make use of Pattern and Matcher class to check for special characters
            Pattern p = Pattern.compile("[~!@#$%^&*()_+{}\\[\\]:;,.<>/?-]");
            Matcher m = p.matcher(newPassword);
            if(m.find()) {
                //update p and m to check for digits
                p = Pattern.compile(".*\\d.*");
                m = p.matcher(newPassword);
                if(m.find()) {
                    //update p and m to check for uppercase letters
                    p = Pattern.compile(".*[A-Z].*");
                    m = p.matcher(newPassword);
                    if(m.find()) {
                        //now we can flip the flag
                        meetsCriteria = true;
                    } //endif
                } //endif
            } //endif
        } //endif
        return meetsCriteria;
    } //end method validateNewPassword
} //end class LogInController
```