# Assignment Objective

Design, plan, and implement a program that validates a user's proposed password against a set of security rules.

# Core Tasks

## Part 1: Problem Analysis & Design

**1. Understand the Rules**: Your validator must check a candidate `String` password for the following criteria:

- It must be at least 8 characters long.
- It must contain at least one uppercase letter ( `A-Z` ).
- It must contain at least one lowercase letter ( `a-z` ).
- It must contain at least one digit ( `0-9` ).
- It must contain at least one special character from this set: `! @ # $ % & * ?`.

**2. Decompose the Problem (and then use Abstraction)**:

A key validation task is counting how many characters in the password belong to a specific group (uppercase letters, lower case letters, digits, special characters). Design a subprocess/method called `charCountInString` that conforms to the design requirements:

- Parameters to this method: a `String`, the password, and an array of characters ( `char[]` ) to count.
- Algorithm: the method will need to loop through the password and count how many characters in the `password` match any character in the provided character array parameter.
- Return: the count of matching characters (an integer, type `int`).

The `charCountInString` method is then to be called by a subprocess/method – call this method `validatePassword` – that will pass the appropriate parameters to check that the password follows each of the rules mentioned above, other than the first rule that verifies the password length, which will not need to call method `charCountInString`. The `validatePassword` method should:

- use either a chained `if` statement or sequential `if` statements to test the different rules
- print a message to the console for either (a) the first rule that it finds has failed, or (b) all rules that have failed.
- return `true` if the password is valid (all rules have been followed), otherwise return `false`.

**4. Documentation**: draw separate flowcharts for the main process, the logic inside the `charCountInString` subprocess, and the logic inside the `validatePassword` subprocess. You may decompose the problem into even smaller subprocesses, if that makes sense for your design. Some details may be left out of the flowcharts (such as a print method), if the coding implementation will be simple and obvious.

## Part 2: Implementation and Testing

You will build the program in two phases, focusing on reusability.

### *Phase A: Write and Test* `charCountInString`

Code the `charCountInString` method exactly as you designed and documented it with your flowchart.

***Hint***: you will need to extract single characters from the password string using one of the string methods, perhaps method `charAt` or method `toCharArray`. You can search for descriptions of these methods online and decide which you feel will result in the easier code to understand. I suggest the Oracle website as a reference for Java methods.

After completing the method and before you write method `validatePassword`, which will call this method, test the `charCountInString` method in isolation in a simple way – i.e.: hardcode a test string and character array, print the result.

Pearson Pseudocode example of testing the count of digits by method `charCountInString`:

```
SET digits TO [ '0', '1', '2', '3', '4', '5', '6', '7', '8', '9' ]
SET password TO "pa55w0rd"
SET charCount TO charCountInString(password, digits)
SEND charCount TO DISPLAY
```

You should test more than one password to ensure your code works as expected. Remember to test the extreme cases: when the password contains no characters at all, when there are no digits in the password, and when all characters in the password are digits.

### *Phase B: Write* `validatePassword`

Main program logic that will provide some simple test cases for the method `validatePassword` is given in ***Phase C***. As `validatePassword` calls method `charCountInString`, that method will be tested again, partially and indirectly. However, it will be difficult to debug the entire program if `charCountInString` is not working properly prior to implementing method `validatePassword`.

Study the main program. You should see that you will need to have method `validatePassword` in a class named `PasswordValidator`. The main program will print out a message if the password is found to be valid, for example, for the password "`V@1idP@ssW0rd`", the given main program will print:

```
Password "V@1idP@ssW0rd" is a valid password.
```

There is no output for any invalid password. The output for invalid passwords should come from method `validatePassword` itself.

Code the `validatePassword` method. When run with the main program test code given in `PasswordValidatorTester`, the output should be exactly the same as, or somehow better than, the example output shown.

## *Phase C: Test and Debug the* `PasswordValidator` *Code*

Run the `PasswordValidatorTester` and debug your code until the output is exactly as shown.

Java class with test cases for method `PasswordValidator.validatePassword`:

```
public class PasswordValidatorTester {
    private static String[] passwords = {
        "",
        "$h0rT",         // fail: not eight characters in length
        "n0upperca$e",   // fail: no upper case
        "N0LOWERC@SE",   // fail: no lower case
        "noD!gits",      // fail: no digits
        "noSpecia1",     // fail: no special characters
        "V@1idP@ssW0rd", // valid password
        "1234567890aA@", // valid password
        "validpass!1L",  // valid password
    };
    public static void main(String[] args) {
        PasswordValidator.printCharArrays();
        for(String password: passwords) {
            boolean valid =
                PasswordValidator.validatePassword(password);
            if(valid) {
                System.out.println("Password \"" + password +
                    "\" is a valid password.");
            }
        }
    }
}
```

Example output of `PaswordValidatorTester`:

```
Password "" is too short.
Password "$h0rT" is too short.
Password "n0upperca$e" contains no upper case letters.
Password "N0LOWERC@SE" contains no lower case letters.
Password "noD!gits" contains no digits.
Password "noSpecia1" does not contain a special character.
  Special characters: [!, @, #, $, %, &, *, ?]
Password "V@1idP@ssW0rd" is a valid password.
Password "1234567890aA@" is a valid password.
Password "validpass!1L" is a valid password.
```

## Part 4: Going Beyond

You can optionally do these additional mini-projects related to this project:

- update the code such that you declare a number of constants at the top of the class, each constant with the minimum number of characters of the specified type. For example, if we wished the code to enforce a minimum of 3 special characters, we would define a constant at the top of the class with code such as:
  ```
  public static final int MINIMUM_SPECIALCHARS = 3;
  ```
  For this, you should probably also modify the error messages to inform the user how many characters of that type the password should have.
- Write an additional method `passwordStrength` that returns an integer value that represents how strong the password is. The integer value is to represent the number of tests that are passed. Failing all tests will return zero ( `0` ), and for each test passed, an additional point will be added. Write test code that will test the method thoroughly.
- Write and test a methods `countVowels` and `countConsonants`, each of which call the `charCountInString` method and return the number of vowels or consonants, respectively, in a `String`.

# Minimum Completion Checklist

✅ Complete flowcharts for the algorithms in `charCountInString` and `validatePassword`.

✅ `charCountInString` method is implemented, tested, and works correctly.

✅ `validatePassword` method is implemented, tested using the given `PasswordValidatorTester` class, and works correctly.

✅ Code is neatly formatted and uses meaningful variable names.

If you attempt any of the ***Going Beyond*** code challenges, be sure to save and be able to show the results given in the ***Minimum Requirements Checklist***.