Handling User Authentication

About :

Set up a standalone project to do unit testing of the user authentication class which is used in the main web application. The objective is to create a JUnit class that will test all aspects of the authentication class.

Installation Guide:

1. GitHub link: https://github.com/baljeet-singh97/JAVA-Projects/tree/main/Phase%203/userAuthentication

2. Download the entire project as Zip in local system.

3. import the project in Eclipse IDE

Code description

AuthenticationUser.java

Created a simple return type method and defined an email id by default for testing purpose

```
public String username()
        {
                String email = "thealex@gmail.com";
                return email;
        }
```

Created a simple return type method for password and defined an password by default here also.

```
public String paswd()
        {
                String password = "helloiamalex";
                return password;
        }
```

When these function will be called from Junit test method they will return the email and password to the Junit and there testing will be performed on email and password.

AuthTest.java

The @BeforeEach annotation is one of the test lifecycle methods and is the replacement of @Before annotation in JUnit 4. By default, the test methods will be executed in the same thread as @BeforeEach annotated method. Defined this here so in after each defining authU as null so any garbage values will be removed and programm will be having no garbage values.

```
@BeforeEach
        public void setup() {
                authU= new AuthenticationUser();
                System.out.println("Authentication User main class inititated");
        }
```

@AfterEach annotation is used to signal that the annotated method should be executed after each @Test @RepeatedTest, @ParameterizedTest, or @TestFactory methods in the current class. and here nullifying the object after successful completion of testing.

```
@AfterEach
        public void tearDown() {
                authU=null;
                System.out.println("Class Closed");
        }
```

checkUser test case getting the original value from the object and using assertEquals method to check if the email id is correct or not.

```
@Test
        public void checkUser() {
                assertEquals("thealex@gmail.com", authU.username());
        }
```

CheckUserNull test case, using assertNotNull method to check if the username null or not.

```
@Test
        public void checkUserNull()
        {
                assertNotNull(authU.username());
        }
```

CheckPass test case getting password using the object and checking the password with original password.

```
@Test
        public void checkPass() {
                assertEquals("helloiamalex", authU.paswd());
        }
```

CheckPassNull test case checking that password should not be null

```
@Test
        public void checkPassNull()
        {
                assertNotNull(authU.paswd());
        }
```

Defined Regx method to check if the email id enterd by user is valid email id or not.

```
@Test
        public void checkUserRegx()
        {
                String email = authU.username();
                String regex = "^(.+)@(.+)$";

                Pattern pattern = Pattern.compile(regex);
                Matcher matcher = pattern.matcher(email);



                assertEquals(true, matcher.matches());
        }
```