2022.8

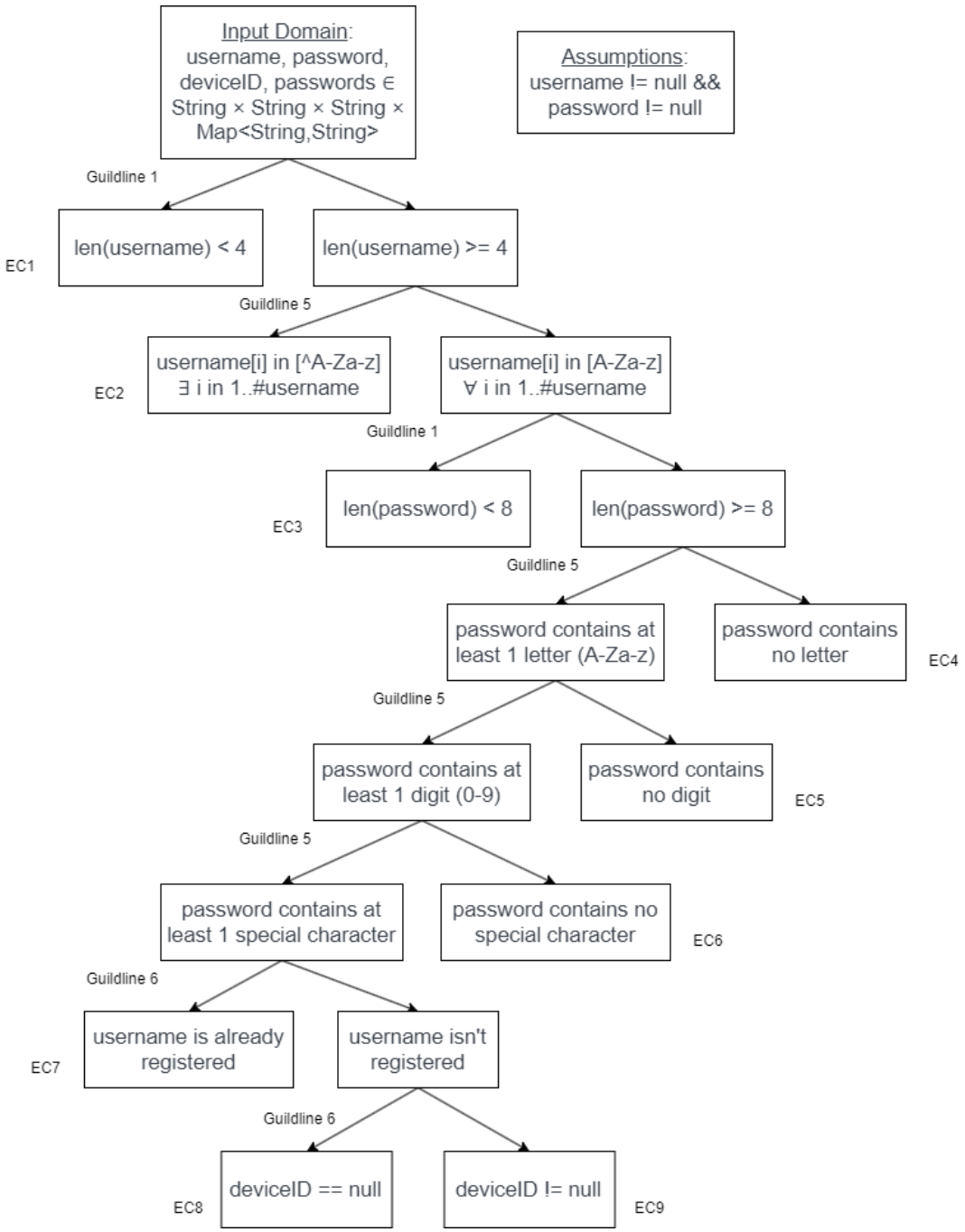
SWEN90006 Security & Software Testing

ASSIGNMENT 1

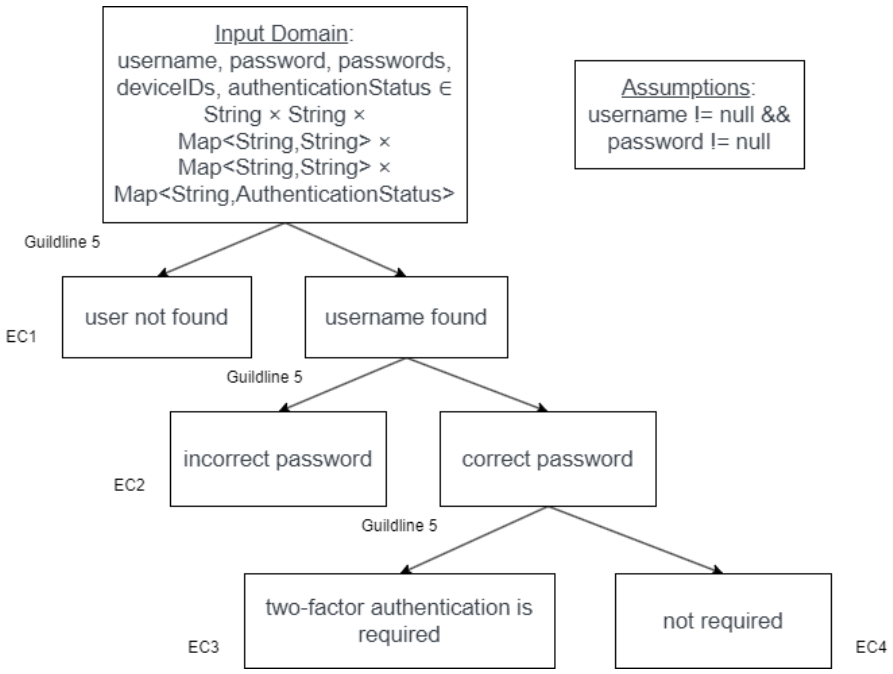
ZHEN CAI (1049487)

1 EQUIVALENCE PARTITIONING

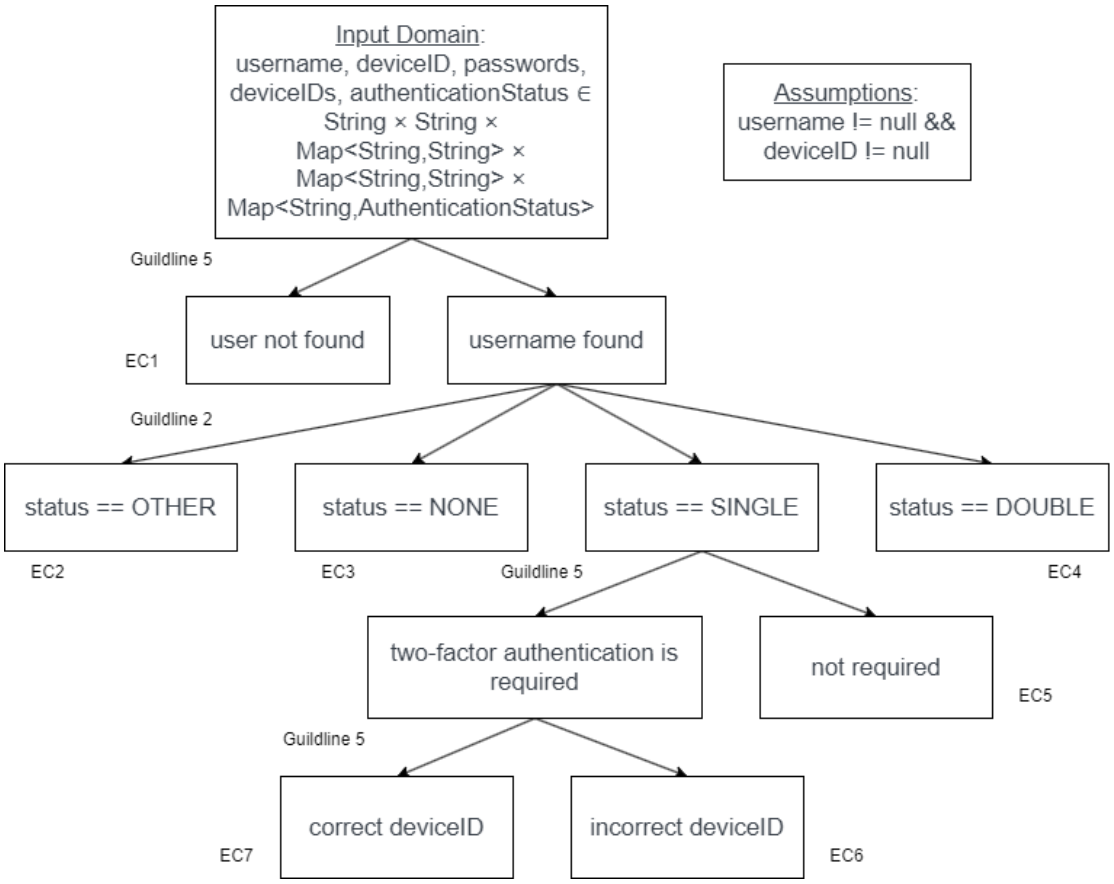
1.1 REGISTER



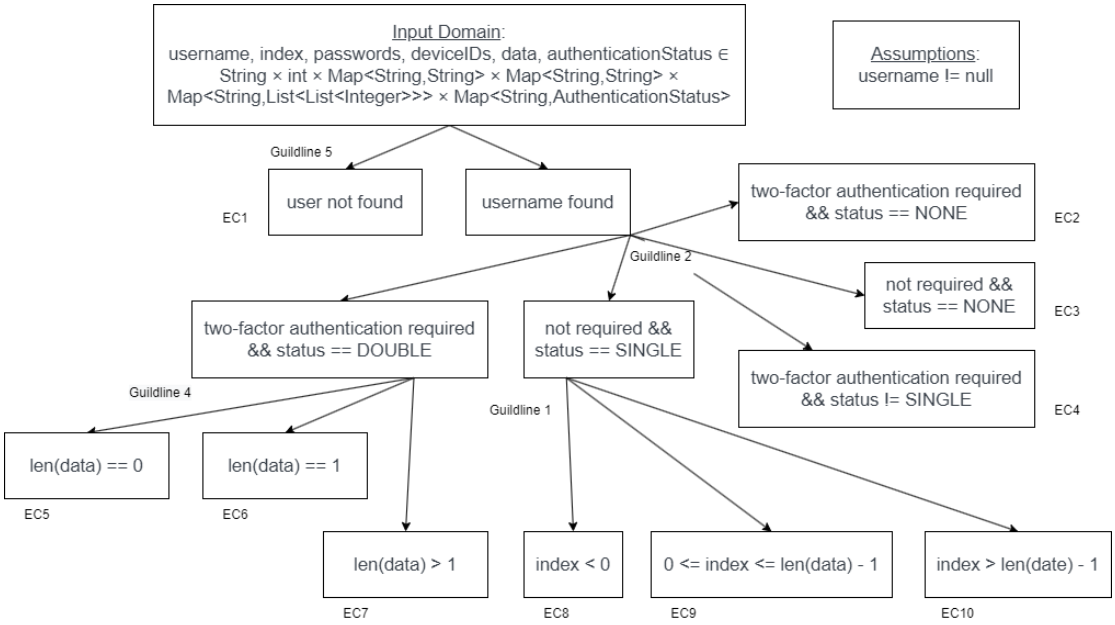
1.2 LOGIN



1.3 RESPONDTOPUSHNOTIFICATION



1.4 GETDATA



1.5 DO YOUR SET OF EQUIVALENCE CLASSES COVER THE INPUT SPACE?

Yes, they do.

Test template trees suggests a hierarchical way to break the testable input domain into a reasonable number of equivalence classes. The partitioning is 1) disjoint, and 2) child nodes inherit parent's conditions plus some of their own, where siblings can combine to cover their parent. In addition, when describing the input domain, We have also considered all instance variables that are involved in the control flow conditions.

2 JUNIT TEST DRIVER FOR EQUIVALENCE PARTITIONING

See Appendix A.

3 BOUNDARY-VALUE ANALYSIS

Note that we assume there's an account named "UserNameA" with password "Password1!" and deviceID "", registered before any test runs. This account currently has no data records. NONE, SINGLE, DOUBLE are just shorthand for AuthenticationStatus.NONE, ...

3.1 REGISTER

	On/Off-points	Testcases	Exp. Results
EC1	on: len(username) = 4 off: len(username) = 3	register("you", "abc123@{", null)	InvalidUsernameException
EC2	on: username contains only alphabets off: username contains 1 non-alphabet	register("@you", "abc123@{", null) register("y[ou", "abc123@{", null) register("yo`u", "abc123@{", null) register("you{", "abc123@{", null)	
EC3	on: len(password) = 8 off: len(password) = 7	register("student", "abc123@", null)	InvalidPasswordException
EC4	on: password contains 1 alphabet off: password contains no alphabet	[EC4 off-points + EC6 on-points] register("student", "@0123459", null) register("student", "0678[`{9", null)	
EC5	on: password contains 1 digit off: password contains no digit	[EC5 off-points + EC4 on-points] register("student", "/:A!@[`{", null) register("student", "/:[`{Zaz", null)	
EC6	on: password contains 1 special character off: password contains no special character	[EC6 off-points + EC5 on-points] register("student", "0AaaZZzz", null) register("student", "99999999A", null)	
EC7	on: username already registered off: username does not exist	register("student", "abc123@{", null) and try to register again	DuplicateUserException
EC8	on: deviceID == null	register("student", "abc123@{", null)	Successfully registered
EC9	off: deviceID != null	register("student", "abc123@{", "iphone$$$")	

Also considers ASCII Boundaries

3.2 LOGIN

	On/Off-points	Testcases	Exp. Results
EC1	on: username found off: username does not exist	login("student", "abc123@{")	NoSuchUserException Return NONE
EC2	on: password, username match	login("UserNameA", "Password2!")	IncorrectPasswordException

	off: they do not match		Return NONE
EC3	on: two-factor authentication enabled by registering a device off: no device added	login(“UserNameA”, “Password1!”)	Return SINGLE
EC4		register(“student”, “abc123@{”, null), then login(“student”, “abc123@{”)	

3.3 RESPONDTOPUSHNOTIFICATION

	On/Off-points	Testcases	Exp. Results
EC1	on: username found off: username does not exist	respondToPushNotification(“student”, “”)	NoSuchUserException
EC2	on: { x | x ∈ [NONE, SINGLE, DOUBLE, ~~OTHER~~]} off: not x	~~No direct write access is given to authenticationStatus, so OTHER may be inserted and fall into untestable~~	
EC3		respondToPushNotification("UserNameA", “”) before login	Return NONE
EC4		respondToPushNotification(“UserNameA”, “”) for the second time after we successfully login and authenticate first	Return DOUBLE
EC5	on: two-factor authentication required by the account off: not required	register(“student”, “abc123@{”, “iphone$$$”), login, then respondToPushNotification(“student”, “iphone$$$”)	Return SINGLE
EC6	on: the correct device is responding to the push off: with an incorrect device	Login, then respondToPushNotification(“UserNameA”, “wrong-device”)	IncorrectDeviceIDException Return SINGLE
EC7		Login, then respondToPushNotification(“UserNameA”, “”)	Return DOUBLE

3.4 GETDATA

	On/Off-points	Testcases	Exp. Results
EC1	on: username found off: username does not exist	getData(“student”, 0)	NoSuchUserException

...

EC2	on: { x | x ∈ [2FA enabled + NONE, 2FA enabled + SINGLE, 2FA enabled + DOUBLE, 2FA disabled + NONE, 2FA disabled + SINGLE ~~2FA disabled + DOUBLE (untestable, I don't make it a separate EP for this time)~~]} off: not x	getData("UserNameA", 0) before login	UnauthenticatedUserException
EC3		register("student", "abc123@{", "iphone$$$"), then getData("student", 0) before login	
EC4		getData("UserNameA", 0) after login but before responding to the push notification	
EC5	on: len(data) == 0 off: len(data) == ~~-1~~, 1	Become 2FA enabled + DOUBLE getData("UserNameA", 0)	IndexOutOfBoundsException
EC6	on: len(data) == 1 off: len(data) == 0, 2	Become 2FA enabled + DOUBLE addData("UserNameA", [-1]) getData("UserNameA", 0)	Return [-1]
EC7	on: len(data) == 1 off: len(data) == 2	Become 2FA enabled + DOUBLE addData("UserNameA", []) addData("UserNameA", [-1, 4]) getData("UserNameA", 1)	Return [-1, 4]
EC8	on: index == 0 off: index == -1	Become 2FA disabled + SINGLE addData("student", []) getData("student", -1)	IndexOutOfBoundsException
EC9	on: index == 0, len(data)-1 off: index == -1, len(data)	Become 2FA disabled + SINGLE addData("student", []) getData("student", 0)	Return []
EC10	on: index == len(data)-1 off: index == len(data)	Become 2FA disabled + SINGLE addData("student", []) getData("student", 1)	IndexOutOfBoundsException

4 JUNIT TEST DRIVER FOR BOUNDARY-VALUE ANALYSIS

See Appendix B.

5 MULTIPLE-CONDITION COVERAGE

* Let us assume that the False case of for-each loop structure means that execution path exits the loop, and True means it enters the loop.

5.1 REGISTER

(BVA #4 means both #4.1 and #4.2 in our test suites; similarly applies to #2/5/6.)

	Conditions	Comb.	=T/F?	Coved by which EP Testcase	Coved by which BVA Testcase
1.	if (passwords.containsKey(username))	T	T	7	
2.		F	F	1-6, 8, 9	
3.	else if (username.length() <	T	T	1	
4.	MINIMUM_USERNAME_LENGTH)	F	F	2-6, 8, 9	
5.	else if (password.length() <	T	T	3	
6.	MINIMUM_PASSWORD_LENGTH)	F	F	2, 4-6, 8, 9	
7.	for (char c : username.toCharArray())	T	T	2, 4-6, 8, 9	
8.		F	F	4-6, 8, 9	
9.	if (!('a' <= c && c <= 'z' || 'A' <= c && c <= 'Z'))	~~TTTT~~	~~F~~		
10.		TTTF	F	4-6, 8, 9	2, 4-6, 8, 9
11.		~~TTFT~~	~~F~~		
12.		~~TTFF~~	~~F~~		
13.		~~TFTT~~	~~F~~		
14.		TFTF	T		2.4
15.		~~TFFT~~	~~T~~		
16.		~~TFFF~~	~~T~~		
17.		FTTT	F		2
18.		FTTF	T		2.2, 2.3
19.		FTFT	T	2	2.1
20.		~~FTFF~~	~~T~~		
21.		~~FFTT~~	~~F~~		
22.		~~FFTF~~	~~T~~		
23.		~~FFFT~~	~~T~~		
24.		~~FFFF~~	~~T~~		
25.	for (char c : password.toCharArray())	T	T	4-6, 8, 9	
26.		F	F	4-6, 8, 9	
27.	if ('a' <= c && c <= 'z' || 'A' <= c && c <= 'Z')	~~TTTT~~	~~T~~		
28.		TTTF	T	5, 6, 8, 9	5.2, 6.1, 8, 9
29.		~~TTFT~~	~~T~~		

30.		~~TTFF~~	~~T~~		
31.		~~TFTT~~	~~T~~		
32.		TFTF	F		4.2, 5, 8, 9
33.		~~TFFT~~	~~F~~		
34.		~~TFFF~~	~~F~~		
35.		FTTT	T	6, 8, 9	5, 6
36.		FTTF	F		4.2, 5
37.		FTFT	F	4-6, 8, 9	
38.		~~FTFF~~	~~F~~		
39.		~~FFTT~~	~~T~~		
40.		~~FFTF~~	~~F~~		
41.		~~FFFT~~	~~F~~		
42.		~~FFFF~~	~~F~~		
43.	else if ('0' <= c && c <= '9')	TT	T	4, 6, 8, 9	
44.		TF	F	4, 5, 8, 9	
45.		FT	F	4	5
46.		~~FF~~	~~F~~		
47.	if (!(letter && digit && special))	TTT	F	8, 9	
48.		TTF	T	6	
49.		TFT	T	5	
50.		TFF	T		
51.		FTT	T	4	
52.		FTF	T		
53.		FFT	T		
54.		~~FFF~~	~~T~~		
55.	if (deviceID != null)	T	T	9	
56.		F	F	8	

With equivalence partitioning, 24 condition combinations are met, and 32 = 24 infeasible + 8 feasible are not met. Within the 8 cases, 5 are derived from character ASCII boundaries, and 3 others are those uncovered False combinations of “letter && digit && special” where I had tested only combinations of 1 False and 2 True.

Coverage_Score_EP = 24 / 56 = 42.86%

Boundary-value analysis considers more about ASCII boundaries. It provides more coverage on #14, 17, 18, 36, 32, and we have

Coverage_Score_BVA = 29 / 56 = 51.79%

5.2 LOGIN

	Conditions	Comb.	=T/F?	Coved by which EP Testcase	Coved by which BVA Testcase
1	if (!isUser(username))	T	T	1	
2		F	F	2-4	
3	else if (!passwords.get(username).equals(password))	T	T	2	
4		F	F	3, 4	
5	if (checkUsernamePassword(username, password))	T	T	3, 4	
6		F	F		
7	if (deviceIDs.get(username) != null)	T	T	3	
8		F	F	4	

For both test suites, #6 is not covered because the inner function never return False before the exception arises and the program should stop first.

Coverage_Score_EPBVA = 7 / 8 = 87.50%

5.3 RESPONDTOPUSHNOTIFICATION

	Conditions	Comb.	=T/F?	Coved by which EP Testcase	Coved by which BVA Testcase
1	if (!isUser(username))	T	T	1	
2		F	F	3-7	
3	else if (authenticationStatus.get(username) == AuthenticationStatus.SINGLE)	T	T	5-7	
4		F	F	3, 4	
5	if (deviceIDs.get(username) != null && deviceIDs.get(username) != deviceID)	TT	T	6	
6		TF	F	7	
7		FT	F	5	
8		~~FF~~	~~F~~		
9	else if (deviceIDs.get(username) != null)	T	T	7	
10		F	F	5	

For both test suites, #8 is not covered because deviceIDs.get(username) == null && deviceIDs.get(username) == deviceID implies deviceID == null, but this contradicts the assumption that deviceID != null therefore is infeasible.

Coverage_Score_EPBVA = 9 / 10 = 90.00%

5.4 GETDATA

	Conditions	Comb.	=T/F?	Coved by which EP Testcase	Coved by which BVA Testcase
1	if (!isUser(username))	T	T	1	
2		F	F	2-10	
3	else if ((deviceIDs.get(username) == null && authenticationStatus.get(username) == AuthenticationStatus.SINGLE) || deviceIDs.get(username) != null && authenticationStatus.get(username) == AuthenticationStatus.DOUBLE))	~~TTTT~~	~~T~~		
4		~~TTTF~~	~~T~~		
5		~~TTFT~~	~~T~~		
6		TTFF	T	8-10	
7		~~TFTT~~	~~T~~		
8		~~TFTF~~	F		
9		~~TFFT~~	~~F~~		
10		TFFF	F	3	
11		~~FTTT~~	~~T~~		
12		FTTF	F	4	
13		~~FTFT~~	~~F~~		
14		~~FTFF~~	~~F~~		
15		FFTT	T	5-7	
16		FFTF	F	2	
17		~~FFFT~~	~~F~~		
18		~~FFFF~~	~~F~~		
19	if (!isAuthenticated(username))	T	T	2-4	
20		F	F	5-10	

For both test suites, 9 condition combinations are met, and 11 infeasible are not met.

Coverage_Score_EPBVA = 9 / 20 = 45.00%

6 MUTATION SELECTION

See my code in programs/mutant-*/swen90006/mfa/MFA.java.

7 COMPARISON

The boundary-value analysis generates more testcases around the ASCII boundaries of digits and alphabets as its on/off points, therefore has coverage scores >= that scores from equivalence partitioning method.

	register	login	respondToPushNotification	getData
EP	42.86%	87.5%	90%	45%
BVA	51.79%	87.5%	90%	45%

There are a few numbers of execution paths that go through some specific condition combinations are infeasible and input can also be untestable; and we can observe for example the 45% coverage for getData function, though our test suites cover all feasible cases.

Both are important block-box testing approaches. Equivalence partitioning creates equivalence classes, boundary-value analysis suggests a way to test against boundary shifts and hopefully is more likely to select better testcases for detecting faults.

In addition, I write 5 mutants of the original program. BVA kills 5/5, EP fails to kill #2. Hence it shows that BVA is more effective than EP.

8 APPENDIX

8.1 A

```
package swen90006.mfa;

import org.junit.*;
import static org.junit.Assert.*;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class PartitioningTests
{
    protected MFA mfa;

    //Any method annotated with "@Before" will be executed before each test,
    //allowing the tester to set up some shared resources.
    @Before public void setUp()
    throws DuplicateUserException, InvalidUsernameException,
InvalidPasswordException
    {
        mfa = new MFA();
        mfa.register("UserNameA", "Password1!", "");
    }

    //Any method annotated with "@After" will be executed after each test,
    //allowing the tester to release any shared resources used in the setup.
    @After public void tearDown()
    {
    }

    /*
        ////////////////////////////////////////////////////////////////////////
        //////////////////  register equivalence partitioning  //////////////////
        ////////////////////////////////////////////////////////////////////////

    */
    @Test(expected = InvalidUsernameException.class)
    public void registerEC1() throws Throwable
    {
```

```
        String username = "me";
        String password = "@un1Me16";
        String deviceID = null;
        mfa.register(username, password, deviceID);
        assertFalse(mfa.isUser(username));
    }

    @Test(expected = InvalidUsernameException.class)
    public void registerEC2() throws Throwable
    {
        String username = "\n \\\"a[-1]";
        String password = "@un1Me16";
        String deviceID = null;
        mfa.register(username, password, deviceID);
        assertFalse(mfa.isUser(username));
    }

    @Test(expected = InvalidPasswordException.class)
    public void registerEC3() throws Throwable
    {
        String username = "student";
        String password = "@u0M";
        String deviceID = null;
        mfa.register(username, password, deviceID);
    }

    @Test(expected = InvalidPasswordException.class)
    public void registerEC4() throws Throwable
    {
        String username = "student";
        String password = "@2022.8.18";
        String deviceID = null;
        mfa.register(username, password, deviceID);
    }

    @Test(expected = InvalidPasswordException.class)
    public void registerEC5() throws Throwable
    {
        String username = "student";
        String password = "@unimelb";
        String deviceID = null;
        mfa.register(username, password, deviceID);
    }
```

```java
@Test(expected = InvalidPasswordException.class)
public void registerEC6() throws Throwable
{
    String username = "student";
    String password = "ATun1me16";
    String deviceID = null;
    mfa.register(username, password, deviceID);
}

@Test(expected = DuplicateUserException.class)
public void registerEC7() throws Throwable
{
    String username = "student";
    String password = "@un1Me16";
    String deviceID = null;
    mfa.register(username, password, deviceID);
    mfa.register(username, password, deviceID);
    assertTrue(mfa.isUser(username));
}

@Test
public void registerEC8() throws Throwable
{
    String username = "student";
    String password = "@un1Me16";
    String deviceID = null;
    mfa.register(username, password, deviceID);
    assertTrue(mfa.isUser(username));
}

@Test
public void registerEC9() throws Throwable
{
    String username = "student";
    String password = "@un1Me16";
    String deviceID = "iphone$$$";
    mfa.register(username, password, deviceID);
    assertTrue(mfa.isUser(username));
}

/*
    ////////////////////////////////////////////////////////////////////////
    //////////////////  login equivalence partitioning  ////////////////////
    ////////////////////////////////////////////////////////////////////////
```

```
*/
@Test(expected = NoSuchUserException.class)
public void loginEC1() throws Throwable
{
    String username = "student";
    String password = "@un1Me16";
    MFA.AuthenticationStatus status = mfa.login(username, password);
    assertEquals(MFA.AuthenticationStatus.NONE, status);
}

@Test(expected = IncorrectPasswordException.class)
public void loginEC2() throws Throwable
{
    String username = "UserNameA";
    String password = "Password2!";
    MFA.AuthenticationStatus status = mfa.login(username, password);
    assertEquals(MFA.AuthenticationStatus.NONE, status);
}

@Test
public void loginEC3() throws Throwable
{
    String username = "UserNameA";
    String password = "Password1!";
    MFA.AuthenticationStatus status = mfa.login(username, password);
    assertEquals(MFA.AuthenticationStatus.SINGLE, status);
}

@Test
public void loginEC4() throws Throwable
{
    String username = "student";
    String password = "@un1Me16";
    mfa.register(username, password, null);
    MFA.AuthenticationStatus status = mfa.login(username, password);
    assertEquals(MFA.AuthenticationStatus.SINGLE, status);
}

/*
    ////////////////////////////////////////////////////////////////////////
    ////////  respondToPushNotification equivalence partitioning  /////////
    ////////////////////////////////////////////////////////////////////////
```

```
    */
    @Test(expected = NoSuchUserException.class)
    public void respondToPushNotificationEC1() throws Throwable
    {
        String username = "student";
        String deviceID = "";
        mfa.respondToPushNotification(username, deviceID);
    }

    /**
     * Cannot test respondToPushNotificationEC2 here,
     * because we have no write access to the private variable
authenticationStatus
     */

    @Test
    public void respondToPushNotificationEC3() throws Throwable
    {
        String username = "UserNameA";
        String deviceID = "";
        MFA.AuthenticationStatus status =
mfa.respondToPushNotification(username, deviceID);
        assertEquals(MFA.AuthenticationStatus.NONE, status);
    }

    @Test
    public void respondToPushNotificationEC4() throws Throwable
    {
        String username = "UserNameA";
        String deviceID = "wrong-device";
        mfa.login(username, "Password1!");
        mfa.respondToPushNotification(username, "");
        MFA.AuthenticationStatus status =
mfa.respondToPushNotification(username, deviceID);
        assertEquals(MFA.AuthenticationStatus.DOUBLE, status);
    }

    @Test
    public void respondToPushNotificationEC5() throws Throwable
    {
        String username = "student";
        String deviceID = "fake-device";
        mfa.register(username, "@un1Me16", null);
        mfa.login(username, "@un1Me16");
```

```
        MFA.AuthenticationStatus status =
mfa.respondToPushNotification(username, deviceID);
        assertEquals(MFA.AuthenticationStatus.SINGLE, status);
    }

    @Test(expected = IncorrectDeviceIDException.class)
    public void respondToPushNotificationEC6() throws Throwable
    {
        String username = "UserNameA";
        String deviceID = "wrong-device";
        mfa.login(username, "Password1!");
        MFA.AuthenticationStatus status =
mfa.respondToPushNotification(username, deviceID);
        assertEquals(MFA.AuthenticationStatus.SINGLE, status);
    }

    @Test
    public void respondToPushNotificationEC7() throws Throwable
    {
        String username = "UserNameA";
        String deviceID = "";
        mfa.login(username, "Password1!");
        MFA.AuthenticationStatus status =
mfa.respondToPushNotification(username, deviceID);
        assertEquals(MFA.AuthenticationStatus.DOUBLE, status);
    }

    /*
        ////////////////////////////////////////////////////////////////////////
        /////////////////  getData equivalence partitioning  /////////////////
        ////////////////////////////////////////////////////////////////////////

    */
    @Test(expected = NoSuchUserException.class)
    public void getDataEC1() throws Throwable
    {
        String username = "student";
        int index = 0;
        mfa.getData(username, index);
    }

    @Test(expected = UnauthenticatedUserException.class)
    public void getDataEC2() throws Throwable
    {
```

```
        String username = "UserNameA";
        int index = 0;
        mfa.getData(username, index);
        assertFalse(mfa.isAuthenticated(username));
    }

    @Test(expected = UnauthenticatedUserException.class)
    public void getDataEC3() throws Throwable
    {
        String username = "student";
        int index = 0;
        mfa.register(username, "@un1Me16", null);
        mfa.getData(username, index);
        assertFalse(mfa.isAuthenticated(username));
    }

    @Test(expected = UnauthenticatedUserException.class)
    public void getDataEC4() throws Throwable
    {
        String username = "UserNameA";
        int index = 0;
        mfa.login(username, "Password1!");
        mfa.getData(username, index);
        assertFalse(mfa.isAuthenticated(username));
    }

    @Test(expected = IndexOutOfBoundsException.class)
    public void getDataEC5() throws Throwable
    {
        String username = "UserNameA";
        int index = 0;
        mfa.login(username, "Password1!");
        mfa.respondToPushNotification(username, "");
        mfa.getData(username, index);
    }

    @Test
    public void getDataEC6() throws Throwable
    {
        String username = "UserNameA";
        int index = 0;
        List<Integer> data = Arrays.asList(-1);
        mfa.login(username, "Password1!");
        mfa.respondToPushNotification(username, "");
```

```
        mfa.addData(username, data);
        List<Integer> data_ = mfa.getData(username, index);
        assertEquals(data, data_);
    }

    @Test
    public void getDataEC7() throws Throwable
    {
        String username = "UserNameA";
        int index = 0;
        List<Integer> data0 = Arrays.asList(-1,0,2,5,9);
        mfa.login(username, "Password1!");
        mfa.respondToPushNotification(username, "");
        mfa.addData(username, data0);
        for (int i = 0; i < 4; i++) {
            mfa.addData(username, new ArrayList<>());
        }
        List<Integer> data0_ = mfa.getData(username, index);
        assertEquals(data0, data0_);
    }

    @Test(expected = IndexOutOfBoundsException.class)
    public void getDataEC8() throws Throwable
    {
        String username = "student";
        int index = -1;
        List<Integer> data = Arrays.asList(-1,0,2,5,9);
        mfa.register(username, "@un1Me16", null);
        mfa.login(username, "@un1Me16");
        mfa.addData(username, data);
        mfa.getData(username, index);
    }

    @Test
    public void getDataEC9() throws Throwable
    {
        String username = "student";
        int index = 3;
        List<Integer> data3 = Arrays.asList(-1,0,2,5,9);
        mfa.register(username, "@un1Me16", null);
        mfa.login(username, "@un1Me16");
        mfa.addData(username, new ArrayList<>());
        mfa.addData(username, new ArrayList<>());
        mfa.addData(username, new ArrayList<>());
```

```
        mfa.addData(username, data3);
        mfa.addData(username, new ArrayList<>());
        List<Integer> data3_ = mfa.getData(username, index);
        assertEquals(data3, data3_);
    }

    @Test(expected = IndexOutOfBoundsException.class)
    public void getDataEC10() throws Throwable
    {
        String username = "student";
        int index = 5;
        List<Integer> data = Arrays.asList(-1,0,2,5,9);
        mfa.register(username, "@un1Me16", null);
        mfa.login(username, "@un1Me16");
        mfa.addData(username, data);
        mfa.getData(username, index);
    }

}
```

8.2 B

```
package swen90006.mfa;

import java.util.List;
import java.util.ArrayList;
import java.util.Arrays;
import java.nio.charset.Charset;
import java.nio.file.Path;
import java.nio.file.Files;
import java.nio.file.FileSystems;

import org.junit.*;
import static org.junit.Assert.*;

//By extending PartitioningTests, we inherit the tests from that class
public class BoundaryTests extends PartitioningTests
{
    /*
        ////////////////////////////////////////////////////////////////////
        /////////////////  register boundary-value analysis  ////////////////
```

```
    ////////////////////////////////////////////////////////////////////////

*/
@Test(expected = InvalidUsernameException.class)
public void registerBVA1() throws Throwable
{
    String username = "you";
    String password = "abc123@{";
    String deviceID = null;
    mfa.register(username, password, deviceID);
    assertFalse(mfa.isUser(username));
}

@Test(expected = InvalidUsernameException.class)
public void registerBVA2_1() throws Throwable
{
    String username = "@AZaz";
    String password = "abc123@{";
    String deviceID = null;
    mfa.register(username, password, deviceID);
    assertFalse(mfa.isUser(username));
}

@Test(expected = InvalidUsernameException.class)
public void registerBVA2_2() throws Throwable
{
    String username = "A[Zaz";
    String password = "abc123@{";
    String deviceID = null;
    mfa.register(username, password, deviceID);
    assertFalse(mfa.isUser(username));
}

@Test(expected = InvalidUsernameException.class)
public void registerBVA2_3() throws Throwable
{
    String username = "AZ`az";
    String password = "abc123@{";
    String deviceID = null;
    mfa.register(username, password, deviceID);
    assertFalse(mfa.isUser(username));
}

@Test(expected = InvalidUsernameException.class)
```

```java
public void registerBVA2_4() throws Throwable
{
    String username = "AZa{z";
    String password = "abc123@{";
    String deviceID = null;
    mfa.register(username, password, deviceID);
    assertFalse(mfa.isUser(username));
}

@Test(expected = InvalidPasswordException.class)
public void registerBVA3() throws Throwable
{
    String username = "student";
    String password = "abc123@";
    String deviceID = null;
    mfa.register(username, password, deviceID);
}

@Test(expected = InvalidPasswordException.class)
public void registerBVA4_1() throws Throwable
{
    String username = "student";
    String password = "@0123459";
    String deviceID = null;
    mfa.register(username, password, deviceID);
}

@Test(expected = InvalidPasswordException.class)
public void registerBVA4_2() throws Throwable
{
    String username = "student";
    String password = "0678[`{9";
    String deviceID = null;
    mfa.register(username, password, deviceID);
}

@Test(expected = InvalidPasswordException.class)
public void registerBVA5_1() throws Throwable
{
    String username = "student";
    String password = "/:A!@[`{";
    String deviceID = null;
    mfa.register(username, password, deviceID);
}
```

```java
@Test(expected = InvalidPasswordException.class)
public void registerBVA5_2() throws Throwable
{
    String username = "student";
    String password = "/:[`{Zaz";
    String deviceID = null;
    mfa.register(username, password, deviceID);
}

@Test(expected = InvalidPasswordException.class)
public void registerBVA6_1() throws Throwable
{
    String username = "student";
    String password = "0AaaZZzz";
    String deviceID = null;
    mfa.register(username, password, deviceID);
}

@Test(expected = InvalidPasswordException.class)
public void registerBVA6_2() throws Throwable
{
    String username = "student";
    String password = "99999999A";
    String deviceID = null;
    mfa.register(username, password, deviceID);
}

@Test(expected = DuplicateUserException.class)
public void registerBVA7() throws Throwable
{
    String username = "student";
    String password = "abc123@{";
    String deviceID = null;
    mfa.register(username, password, deviceID);
    mfa.register(username, password, deviceID);
    assertTrue(mfa.isUser(username));
}

@Test
public void registerBVA8() throws Throwable
{
    String username = "student";
    String password = "abc123@{";
```

```
        String deviceID = null;
        mfa.register(username, password, deviceID);
        assertTrue(mfa.isUser(username));
    }

    @Test
    public void registerBVA9() throws Throwable
    {
        String username = "student";
        String password = "abc123@{";
        String deviceID = "iphone$$$";
        mfa.register(username, password, deviceID);
        assertTrue(mfa.isUser(username));
    }

    /*
        ////////////////////////////////////////////////////////////////////////
        //////////////////  login boundary-value analysis  //////////////////
        ////////////////////////////////////////////////////////////////////////

    */
    @Test(expected = NoSuchUserException.class)
    public void loginBVA1() throws Throwable
    {
        String username = "student";
        String password = "abc123@{";
        MFA.AuthenticationStatus status = mfa.login(username, password);
        assertEquals(MFA.AuthenticationStatus.NONE, status);
    }

    @Test(expected = IncorrectPasswordException.class)
    public void loginBVA2() throws Throwable
    {
        loginEC2();
    }

    @Test
    public void loginBVA3() throws Throwable
    {
        loginEC3();
    }

    @Test
    public void loginBVA4() throws Throwable
```

```
    {
        String username = "student";
        String password = "abc123@{";
        mfa.register(username, password, null);
        MFA.AuthenticationStatus status = mfa.login(username, password);
        assertEquals(MFA.AuthenticationStatus.SINGLE, status);
    }

    /*
        ////////////////////////////////////////////////////////////////////////
        /////////  respondToPushNotification boundary-value analysis  /////////
        ////////////////////////////////////////////////////////////////////////

    */
    @Test(expected = NoSuchUserException.class)
    public void respondToPushNotificationBVA1() throws Throwable
    {
        respondToPushNotificationEC1();
    }

    /**
     * Cannot test respondToPushNotificationBVA2 here,
     * because we have no write access to the private variable
authenticationStatus
     */

    @Test
    public void respondToPushNotificationBVA3() throws Throwable
    {
        respondToPushNotificationEC3();
    }

    @Test
    public void respondToPushNotificationBVA4() throws Throwable
    {
        String username = "UserNameA";
        String deviceID = "";
        mfa.login(username, "Password1!");
        mfa.respondToPushNotification(username, deviceID);
        MFA.AuthenticationStatus status =
mfa.respondToPushNotification(username, deviceID);
        assertEquals(MFA.AuthenticationStatus.DOUBLE, status);
    }
```

```
    @Test
    public void respondToPushNotificationBVA5() throws Throwable
    {
        String username = "student";
        String deviceID = "iphone$$$";
        mfa.register(username, "abc123@{", null);
        mfa.login(username, "abc123@{");
        MFA.AuthenticationStatus status =
mfa.respondToPushNotification(username, deviceID);
        assertEquals(MFA.AuthenticationStatus.SINGLE, status);
    }

    @Test(expected = IncorrectDeviceIDException.class)
    public void respondToPushNotificationBVA6() throws Throwable
    {
        respondToPushNotificationEC6();
    }

    @Test
    public void respondToPushNotificationBVA7() throws Throwable
    {
        respondToPushNotificationEC7();
    }

    /*
        ////////////////////////////////////////////////////////////////////////
        //////////////////  getData boundary-value analysis  ///////////////////
        ////////////////////////////////////////////////////////////////////////

    */
    @Test(expected = NoSuchUserException.class)
    public void getDataBVA1() throws Throwable
    {
        getDataEC1();
    }

    @Test(expected = UnauthenticatedUserException.class)
    public void getDataBVA2() throws Throwable
    {
        getDataEC2();
    }

    @Test(expected = UnauthenticatedUserException.class)
    public void getDataBVA3() throws Throwable
```

```
{
    String username = "student";
    int index = 0;
    mfa.register(username, "abc123@{", null);
    mfa.getData(username, index);
    assertFalse(mfa.isAuthenticated(username));
}

@Test(expected = UnauthenticatedUserException.class)
public void getDataBVA4() throws Throwable
{
    getDataEC4();
}

@Test(expected = IndexOutOfBoundsException.class)
public void getDataBVA5() throws Throwable
{
    getDataEC5();
}

@Test
public void getDataBVA6() throws Throwable
{
    getDataEC6();
}

@Test
public void getDataBVA7() throws Throwable
{
    String username = "UserNameA";
    int index = 1;
    List<Integer> data0 = new ArrayList<>();
    List<Integer> data1 = Arrays.asList(-1,4);
    mfa.login(username, "Password1!");
    mfa.respondToPushNotification(username, "");
    mfa.addData(username, data0);
    mfa.addData(username, data1);
    List<Integer> data1_ = mfa.getData(username, index);
    assertEquals(data1, data1_);
}

@Test(expected = IndexOutOfBoundsException.class)
public void getDataBVA8() throws Throwable
{
```

```
        String username = "student";
        int index = -1;
        List<Integer> data = new ArrayList<>();
        mfa.register(username, "abc123@{", null);
        mfa.login(username, "abc123@{");
        mfa.addData(username, data);
        mfa.getData(username, index);
    }

    @Test
    public void getDataBVA9() throws Throwable
    {
        String username = "student";
        int index = 0;
        List<Integer> data = new ArrayList<>();
        mfa.register(username, "abc123@{", null);
        mfa.login(username, "abc123@{");
        mfa.addData(username, data);
        List<Integer> data_ = mfa.getData(username, index);
        assertEquals(data, data_);
    }

    @Test(expected = IndexOutOfBoundsException.class)
    public void getDataBVA10() throws Throwable
    {
        String username = "student";
        int index = 1;
        List<Integer> data = new ArrayList<>();
        mfa.register(username, "abc123@{", null);
        mfa.login(username, "abc123@{");
        mfa.addData(username, data);
        mfa.getData(username, index);
    }

}
```