

# Individual Assignment - Regular Expressions JUnit Test

*Name: Sopheanith Ny*

---

## 1. SSN:

```
1  import org.junit.jupiter.api.Test;
2  import static org.junit.Assert.assertFalse;
3  import static org.junit.Assert.assertTrue;
4
5  public class SSNValidationTest {
6
7      @Test
8      void testValidSSNAllDigits() { // Test valid SSN with all digits (123456789)
9          assertTrue(regexValidator.validSSN(theSSN:"123456789"));
10     }
11
12     @Test
13     void testValidSSNDashes() { // Test valid SSN with dashes (123-45-6789)
14         assertTrue(regexValidator.validSSN(theSSN:"123-45-6789"));
15     }
16
17     @Test
18     void testValidSSNSpaces() { // Test valid SSN with spaces (123 45 6789)
19         assertTrue(regexValidator.validSSN(theSSN:"123 45 6789"));
20     }
21
22     @Test
23     void testValidSSNWithEdgeCaseLowest() { // Test valid SSN with edge case (lowest possible SSN 001-01-0001)
24         assertTrue(regexValidator.validSSN(theSSN:"001-01-0001"));
25     }
26
27     @Test
28     void testValidSSNWithEdgeCaseHighest() { // Test valid SSN with edge case (highest possible SSN 999-99-9999)
29         assertTrue(regexValidator.validSSN(theSSN:"999-99-9999"));
30     }
31
32     @Test
33     void testValidSSNWithMidRange() { // Test valid SSN with mid-range numbers (456-78-9012)
34         assertTrue(regexValidator.validSSN(theSSN:"456-78-9012"));
35     }
36
37     @Test
38     void testValidSSNWithDifferentNumbers() { // Test valid SSN with different numbers (212-58-4567)
39         assertTrue(regexValidator.validSSN(theSSN:"212-58-4567"));
40     }
41
42     @Test
43     void testValidSSNWithSpecialCase() { // Test valid SSN with a special case (078-05-1120)
44         assertTrue(regexValidator.validSSN(theSSN:"078-05-1120"));
45     }
46 }
```

```

47     @Test
48     void testInvalidSSNAllZeroes() { // Test invalid SSN with all zeroes (000-00-0000)
49         assertFalse(regexValidator.validSSN(theSSN:"000-00-0000"));
50     }
51
52     @Test
53     void testInvalidSSNStartsWith666() { // Test invalid SSN starting with 666 (666-00-1234)
54         assertFalse(regexValidator.validSSN(theSSN:"666-00-1234"));
55     }
56
57     @Test
58     void testInvalidSSNWithLetters() { // Test invalid SSN with letters (abc-de-ghij)
59         assertFalse(regexValidator.validSSN(theSSN:"abc-de-ghij"));
60     }
61
62     @Test
63     void testInvalidSSNTooManyDigits() { // Test invalid SSN with too many digits (999-99-99999)
64         assertFalse(regexValidator.validSSN(theSSN:"999-99-99999"));
65     }
66
67     @Test
68     void testInvalidSSNTooFewDigits() { // Test invalid SSN with too few digits (123-45-67)
69         assertFalse(regexValidator.validSSN(theSSN:"123-45-67"));
70     }
71
72     @Test
73     void testInvalidSSNWrongSeparators() { // Test invalid SSN with wrong separators (123.45.6789)
74         assertFalse(regexValidator.validSSN(theSSN:"123.45.6789"));
75     }
76
77     @Test
78     void testInvalidSSNWithInvalidAreaNumber() { // Test invalid SSN with invalid area number (900-12-3456)
79         assertFalse(regexValidator.validSSN(theSSN:"900-12-3456"));
80     }
81
82     @Test
83     void testInvalidSSNEmptyString() { // Test invalid SSN with empty string
84         assertFalse(regexValidator.validSSN(theSSN:""));
85     }
86
87     // Extra credit tests for SSN numbering rules
88     @Test
89     public void testInvalidSSNZeroGroup() { // Test invalid SSN with zero group (000-00-0000)
90         assertFalse(regexValidator.validSSN(theSSN:"000-00-0000"));
91     }
92
93     @Test
94     public void testInvalidSSNInvalidPrefix() { // Test invalid SSN with invalid prefix (666-12-3456)
95         assertFalse(regexValidator.validSSN(theSSN:"666-12-3456"));
96     }
97 }
98

```

## 2. US Phone Number:

```
1 import org.junit.jupiter.api.Test;
2 import static org.junit.Assert.assertFalse;
3 import static org.junit.Assert.assertTrue;
4
5 public class PhoneNumberValidationTest {
6
7     @Test
8     void testValidPhoneNumberWithDashes() { // Test valid phone number with dashes (213-456-7890)
9         assertTrue(regexValidator.validPhoneNumber(thePhoneNumber:"213-456-7890"));
10    }
11
12    @Test
13    void testValidPhoneNumberWithDots() { // Test valid phone number with dots (415.456.7890)
14        assertTrue(regexValidator.validPhoneNumber(thePhoneNumber:"415.456.7890"));
15    }
16
17    @Test
18    void testValidPhoneNumberWithSpaces() { // Test valid phone number with spaces (718 456 7890)
19        assertTrue(regexValidator.validPhoneNumber(thePhoneNumber:"718 456 7890"));
20    }
21
22    @Test
23    void testValidPhoneNumberWithParentheses() { // Test valid phone number with parentheses (503) 456-7890
24        assertTrue(regexValidator.validPhoneNumber(thePhoneNumber:"(503) 456-7890"));
25    }
26
27    @Test
28    void testValidPhoneNumberWithCountryCode() { // Test valid phone number with country code (+1 646-456-7890)
29        assertTrue(regexValidator.validPhoneNumber(thePhoneNumber:"+1 646-456-7890"));
30    }
31
32    @Test
33    void testValidPhoneNumberWithoutSeparators() { // Test valid phone number without separators (8184567890)
34        assertTrue(regexValidator.validPhoneNumber(thePhoneNumber:"8184567890"));
35    }
36
37    @Test
38    void testValidPhoneNumberWithDifferentAreaCode() { // Test valid phone number with different area code (707)999-9999
39        assertTrue(regexValidator.validPhoneNumber(thePhoneNumber:"(707)999-9999"));
40    }
41
42    @Test
43    void testValidPhoneNumberWithMidRange() { // Test valid phone number with mid-range area code (212) 555-7890
44        assertTrue(regexValidator.validPhoneNumber(thePhoneNumber:"(212) 555-7890"));
45    }
46}
```

```

47     @Test
48     void testInvalidPhoneNumberTooShort() { // Test invalid phone number that is too short (999-9999)
49         |     assertFalse(regexValidator.validPhoneNumber(thePhoneNumber:"999-9999"));
50     }
51
52     @Test
53     void testInvalidPhoneNumberWithLetters() { // Test invalid phone number with Letters (abcdefghij)
54         |     assertFalse(regexValidator.validPhoneNumber(thePhoneNumber:"abcdefghij"));
55     }
56
57     @Test
58     void testInvalidPhoneNumberWrongSeparator() { // Test invalid phone number with wrong separator (123/456/7890)
59         |     assertFalse(regexValidator.validPhoneNumber(thePhoneNumber:"123/456/7890"));
60     }
61
62     @Test
63     void testInvalidPhoneNumberExtraDigit() { // Test invalid phone number with extra digit (123-456-78901)
64         |     assertFalse(regexValidator.validPhoneNumber(thePhoneNumber:"123-456-78901"));
65     }
66
67     @Test
68     void testInvalidPhoneNumberWithMissingDigit() { // Test invalid phone number with missing digit (123-456-789)
69         |     assertFalse(regexValidator.validPhoneNumber(thePhoneNumber:"123-456-789"));
70     }
71
72     @Test
73     void testInvalidPhoneNumberWithExtraCharacters() { // Test invalid phone number with extra characters ((123) 456-7890
74         |     assertFalse(regexValidator.validPhoneNumber(thePhoneNumber:"(123) 456-7890 x123"));
75     }
76
77     @Test
78     void testInvalidPhoneNumberWithInvalidAreaCode() { // Test invalid phone number with invalid area code (000) 456-7890
79         |     assertFalse(regexValidator.validPhoneNumber(thePhoneNumber:"(000) 456-7890"));
80     }
81
82     @Test
83     void testInvalidPhoneNumberEmptyString() { // Test invalid phone number with an empty string
84         |     assertFalse(regexValidator.validPhoneNumber(thePhoneNumber:""));
85     }
86 }
87

```

### 3. Email Address:

```
1 import org.junit.jupiter.api.Test;
2 import static org.junit.jupiter.api.Assertions.*;
3
4 public class EmailValidationTest {
5
6     @Test
7     void testValidEmailSimple() { //Tests a simple email format
8         assertTrue(regexValidator.validEmail(theEmail:"john.doe@example.com"));
9     }
10
11     @Test
12     void testValidEmailWithNumbers() { //Tests an email containing numbers in the local part
13         assertTrue(regexValidator.validEmail(theEmail:"user123@domain.net"));
14     }
15
16     @Test
17     void testValidEmailWithUnderscore() { //Tests an email containing an underscore
18         assertTrue(regexValidator.validEmail(theEmail:"user_name@example.org"));
19     }
20
21     @Test
22     void testValidEmailWithDash() { //Tests an email containing a hyphen in the local part
23         assertTrue(regexValidator.validEmail(theEmail:"user-name@domain.com"));
24     }
25
26     @Test
27     void testValidEmailWithSubdomain() { //Tests an email with a subdomain
28         assertTrue(regexValidator.validEmail(theEmail:"contact@sub.example.co.uk"));
29     }
30
31     @Test
32     void testValidEmailWithLongTLD() { //Tests an email with a Long top-Level domain (TLD)
33         assertTrue(regexValidator.validEmail(theEmail:"person@company.travel"));
34     }
35
36     @Test
37     void testValidEmailWithCapitalLetters() { //Tests an email with capital letters
38         assertTrue(regexValidator.validEmail(theEmail:"John.Doe@Example.COM"));
39     }
40
41     @Test
42     void testValidEmailWithNumbersAndHyphens() { //Tests an email with numbers and hyphens
43         assertTrue(regexValidator.validEmail(theEmail:"test-email123@company.biz"));
44     }
45 }
```

```

46     @Test
47     void testInvalidEmailMissingAtSymbol() { //Tests an invalid email missing the '@' symbol
48         assertFalse(regexValidator.validEmail(theEmail:"userexample.com"));
49     }
50
51     @Test
52     void testInvalidEmailMultipleAtSymbols() { //Tests an invalid email with multiple '@' symbols
53         assertFalse(regexValidator.validEmail(theEmail:"user@example.com"));
54     }
55
56     @Test
57     void testInvalidEmailStartingWithDot() { //Tests an invalid email starting with a dot
58         assertFalse(regexValidator.validEmail(theEmail:".user@example.com"));
59     }
60
61     @Test
62     void testInvalidEmailEndingWithDot() { //Tests an invalid email ending with a dot before @
63         assertFalse(regexValidator.validEmail(theEmail:"user.@example.com"));
64     }
65
66     @Test
67     void testInvalidEmailNoDomain() { //Tests an invalid email with no domain name
68         assertFalse(regexValidator.validEmail(theEmail:"user@.com"));
69     }
70
71     @Test
72     void testInvalidEmailInvalidCharacters() { //Tests an invalid email containing special characters
73         assertFalse(regexValidator.validEmail(theEmail:"user!name@example.com"));
74     }
75
76     @Test
77     void testInvalidEmailDoubleDots() { //Tests an invalid email with consecutive dots in the local part
78         assertFalse(regexValidator.validEmail(theEmail:"user..name@example.com"));
79     }
80
81     @Test
82     void testInvalidEmailNoTLD() { //Tests an invalid email missing a domain
83         assertFalse(regexValidator.validEmail(theEmail:"user@example"));
84     }
85 }
86

```

#### 4. Full Name:

```

1  import org.junit.jupiter.api.Test;
2  import static org.junit.jupiter.api.Assertions.*;
3
4  public class NameValidationTest {
5      @Test
6      void testValidNameWithoutMiddleInitial() { // Test valid name without middle initial (Smith, John)
7          assertTrue(regexValidator.validName(theName:"Smith, John"));
8      }
9
10     @Test
11     void testValidNameWithMiddleInitial() { // Test valid name with middle initial (Smith, John L)
12         assertTrue(regexValidator.validName(theName:"Smith, John L"));
13     }
14
15     @Test
16     void testValidNameWithDoubleLastName() { // Test valid name with double last name (Smith-Jones, Mary K)
17         assertTrue(regexValidator.validName(theName:"Smith-Jones, Mary K"));
18     }
19
20     @Test
21     void testValidNameWithApostrophe() { // Test valid name with apostrophe in last name (O'Connor, Liam P)
22         assertTrue(regexValidator.validName(theName:"O'Connor, Liam P"));
23     }
24
25     @Test
26     void testValidNameWithMultipleMiddleInitials() { // Test valid name with multiple middle initials (Brown, Sarah J K)
27         assertTrue(regexValidator.validName(theName:"Brown, Sarah J K"));
28     }
29
30     @Test
31     void testValidNameWithHyphenatedFirstName() { // Test valid name with hyphenated first name (Lee, Anne-Marie)
32         assertTrue(regexValidator.validName(theName:"Lee, Anne-Marie"));
33     }
34
35     @Test
36     void testValidNameWithMiddleInitialLowerCase() { // Test invalid name with lowercase middle initial (Smith, John l)
37         assertFalse(regexValidator.validName(theName:"Smith, John l"));
38     }
39
40     @Test
41     void testValidNameWithExtraSpaces() { // Test invalid name with extra spaces around name ( Smith, John )
42         assertFalse(regexValidator.validName(theName:" Smith, John "));
43     }
44
45     @Test
46     void testInvalidNameMissingComma() { // Test invalid name missing a comma (Smith John)
47         assertFalse(regexValidator.validName(theName:"Smith John"));
48     }
49
50     @Test
51     void testInvalidNameWithNumbers() { // Test invalid name containing numbers (Johnson, Mark 2)
52         assertFalse(regexValidator.validName(theName:"Johnson, Mark 2"));
53     }
54
55     @Test
56     void testInvalidNameWithSpecialCharacters() { // Test invalid name with special characters (Doe, @lice)
57         assertFalse(regexValidator.validName(theName:"Doe, @lice"));
58     }
59
60     @Test
61     void testInvalidNameOnlyComma() { // Test invalid name with only a comma (,)
62         assertFalse(regexValidator.validName(theName:","));
63     }
64
65     @Test
66     void testInvalidNameMiddleInitialTooLong() { // Test invalid name with a middle initial too long (Davis, Chris XYZ)
67         assertFalse(regexValidator.validName(theName:"Davis, Chris XYZ"));
68     }
69
70     @Test
71     void testInvalidNameOnlyFirstName() { // Test invalid name with only a first name (John)
72         assertFalse(regexValidator.validName(theName:"John"));
73     }
74
75     @Test
76     void testInvalidNameOnlyLastName() { // Test invalid name with only a last name (Smith,)
77         assertFalse(regexValidator.validName(theName:"Smith,");
78     }
79 }
80

```

## 5. Date in MM-DD-YYYY:

```
1  import org.junit.jupiter.api.Test;
2  import static org.junit.jupiter.api.Assertions.*;
3
4  public class DateValidationTest {
5
6      // Test valid date formats
7      @Test
8      public void testValidDateStandardFormat() { //Tests a standard date format with dashes
9          assertTrue(regexValidator.validate(theDate:"12-31-2023"));
10     }
11
12     @Test
13     public void testValidDateSlashSeparator() { //Tests a standard date format with slashes
14         assertTrue(regexValidator.validate(theDate:"12/31/2023"));
15     }
16
17     @Test
18     public void testValidLeapYearDate() { //Tests a Leap year date
19         assertTrue(regexValidator.validate(theDate:"02-29-2024"));
20     }
21
22     @Test
23     public void testValidEdgeCaseFirstDayOfYear() { //Tests the first day of the year
24         assertTrue(regexValidator.validate(theDate:"01-01-2023"));
25     }
26
27     @Test
28     public void testValidEdgeCaseLastDayOfYear() { //Tests the Last day of the year
29         assertTrue(regexValidator.validate(theDate:"12-31-2023"));
30     }
31
32     @Test
33     public void testValidShortYearFormat() { //Tests a short year format
34         assertTrue(regexValidator.validate(theDate:"12-31-23"));
35     }
36
37     @Test
38     public void testValidMonthWithLeadingZero() { //Tests a month with a Leading zero
39         assertTrue(regexValidator.validate(theDate:"02-15-2024"));
40     }
41
42     @Test
43     public void testValidMonthWithoutLeadingZero() { //Tests a month without a Leading zero
44         assertTrue(regexValidator.validate(theDate:"2-15-2024"));
45     }
46 }
```



```

47 // Test invalid date formats
48 @Test
49 public void testInvalidDateInvalidMonth() { //Tests an invalid month value
50     assertFalse(regexValidator.validate(theDate:"13-01-2023"));
51 }
52
53 @Test
54 public void testInvalidDateInvalidDay() { //Tests an invalid day value
55     assertFalse(regexValidator.validate(theDate:"02-30-2023"));
56 }
57
58 @Test
59 public void testInvalidDateInvalidLeapYear() { //Tests an invalid Leap year date
60     assertFalse(regexValidator.validate(theDate:"02-29-2023"));
61 }
62
63 @Test
64 public void testInvalidDateWrongSeparator() { //Tests an incorrect separator
65     assertFalse(regexValidator.validate(theDate:"12.31.2023"));
66 }
67
68 @Test
69 public void testInvalidDateNonNumeric() { //Tests a date containing non-numeric characters
70     assertFalse(regexValidator.validate(theDate:"12-AB-2023"));
71 }
72
73 @Test
74 public void testInvalidDateEmptyString() { //Tests an empty string
75     assertFalse(regexValidator.validate(theDate:""));
76 }
77
78 @Test
79 public void testInvalidDateMissingComponents() { //Tests a date missing components
80     assertFalse(regexValidator.validate(theDate:"12-2023"));
81 }
82
83 @Test
84 public void testInvalidDateOutOfRange() { //Tests a date with out-of-range values
85     assertFalse(regexValidator.validate(theDate:"00-00-0000"));
86 }
87 }
88

```

6. House Address:

```

1  import org.junit.jupiter.api.Test;
2  import static org.junit.Assert.assertFalse;
3  import static org.junit.Assert.assertTrue;
4
5  public class AddressValidationTest {
6
7      // Test valid addresses
8      @Test
9      public void testValidAddressStandardFormat() { //Tests a standard address format
10         assertTrue(regexValidator.validAddress(theAddress:"123 Main Street"));
11     }
12
13     @Test
14     public void testValidAddressWithAbbreviation() { //Tests abbreviation of street types
15         assertTrue(regexValidator.validAddress(theAddress:"456 Maple Ave"));
16     }
17
18     @Test
19     public void testValidAddressWithFullWordForStreetType() { //Tests full street type name (Boulevard)
20         assertTrue(regexValidator.validAddress(theAddress:"789 Oak Boulevard"));
21     }
22
23     @Test
24     public void testValidAddressWithDirectionalPrefix() { //Tests address with a directional prefix
25         assertTrue(regexValidator.validAddress(theAddress:"321 North Washington Street"));
26     }
27
28     @Test
29     public void testValidAddressWithMultiWordStreetName() { //Tests street name with multiple words
30         assertTrue(regexValidator.validAddress(theAddress:"654 Martin Luther King Avenue"));
31     }
32
33     @Test
34     public void testValidAddressWithApartmentNumber() { //Tests address containing an apartment number
35         assertTrue(regexValidator.validAddress(theAddress:"987 Pine Street Apt 48"));
36     }
37
38     @Test
39     public void testValidAddressWithUnitNumber() { //Tests address containing a unit number
40         assertTrue(regexValidator.validAddress(theAddress:"246 Cedar Road Unit 7"));
41     }
42
43     @Test
44     public void testValidAddressWithDirectionalAndAbbreviation() { //Tests address with a directional prefix and an abbrevi
45         assertTrue(regexValidator.validAddress(theAddress:"135 South Park Blvd"));
46     }
47

```

```

48 // Test invalid addresses
49 @Test
50 public void testInvalidAddressMissingStreetNumber() { //Tests an address missing a street number
51     assertFalse(regexValidator.validAddress(theAddress:"Main Street"));
52 }
53
54 @Test
55 public void testInvalidAddressWithSpecialCharacters() { //Tests an address containing special characters
56     assertFalse(regexValidator.validAddress(theAddress:"123! Main Street#"));
57 }
58
59 @Test
60 public void testInvalidAddressEmptyString() { //Tests an empty string as an address
61     assertFalse(regexValidator.validAddress(theAddress:""));
62 }
63
64 @Test
65 public void testInvalidAddressOnlyNumbers() { //Tests an address that only contains numbers
66     assertFalse(regexValidator.validAddress(theAddress:"12345"));
67 }
68
69 @Test
70 public void testInvalidAddressWithUnrecognizedStreetType() { //Tests an invalid street type that is not recognized
71     assertFalse(regexValidator.validAddress(theAddress:"456 Main Roadway"));
72 }
73
74 @Test
75 public void testInvalidAddressNumericStreetName() { //Tests an invalid numeric street name
76     assertFalse(regexValidator.validAddress(theAddress:"789 123 Street"));
77 }
78
79 @Test
80 public void testInvalidAddressWithExcessiveSpaces() { //Tests an address with excessive spaces between words
81     assertFalse(regexValidator.validAddress(theAddress:"321 Main Street"));
82 }
83
84 @Test
85 public void testInvalidAddressWithSymbols() { //Tests an address containing symbols
86     assertFalse(regexValidator.validAddress(theAddress:"654 Main & Street"));
87 }
88 }
89

```

## 7. City Location:

```

1  import org.junit.jupiter.api.Test;
2  import static org.junit.Assert.assertFalse;
3  import static org.junit.Assert.assertTrue;
4
5  public class LocationValidationTest {
6
7      @Test
8      public void testValidStandardCityStateZip() {
9          assertTrue(regexValidator.validLocation(theLocation:"Seattle, WA 98101")); //Test Valid city, state, zip format
10     }
11
12     @Test
13     public void testValidCityStateZipWithMultiWordCity() {
14         assertTrue(regexValidator.validLocation(theLocation:"San Francisco, CA 94105")); //Test Multi-word city
15     }
16
17     @Test
18     public void testValidCityStateZipWithHyphenatedCity() {
19         assertTrue(regexValidator.validLocation(theLocation:"Ann Arbor, MI 48104")); //Test Hyphenated city
20     }
21
22     @Test
23     public void testValidCityStateZipWithExtendedZip() {
24         assertTrue(regexValidator.validLocation(theLocation:"Portland, OR 97201-1234")); //Test Extended zip code
25     }
26
27     @Test
28     public void testValidCityStateZipWithSpaceInState() {
29         assertTrue(regexValidator.validLocation(theLocation:"New York, NY 10001")); //Test Space in state abbreviation
30     }
31
32     @Test
33     public void testValidCityStateZipLowercaseState() {
34         assertTrue(regexValidator.validLocation(theLocation:"Chicago, IL 60601")); //Test Lowercase state
35     }
36
37     @Test
38     public void testValidCityWithApostrophe() {
39         assertTrue(regexValidator.validLocation(theLocation:"O'Fallon, MO 63366")); //Test Apostrophe in city
40     }
41
42     @Test
43     public void testValidCityStateZipWithAlternateSpacing() {
44         assertTrue(regexValidator.validLocation(theLocation:"Denver, CO 80202")); //Test Alternate spacing
45     }
46

```

```

47     @Test
48     public void testInvalidCityStateZipMissingState() {
49         assertFalse(regexValidator.validLocation(theLocation:"Seattle 98101")); //Test Missing state
50     }
51
52     @Test
53     public void testInvalidCityStateZipInvalidStateAbbreviation() {
54         assertFalse(regexValidator.validLocation(theLocation:"Seattle, XX 98101")); //Test Invalid state abbreviation
55     }
56
57     @Test
58     public void testInvalidCityStateZipWrongOrder() {
59         assertFalse(regexValidator.validLocation(theLocation:"WA Seattle 98101")); //Test Wrong order
60     }
61
62     @Test
63     public void testInvalidCityStateZipMissingZip() {
64         assertFalse(regexValidator.validLocation(theLocation:"Seattle, WA")); //Test Missing zip code
65     }
66
67     @Test
68     public void testInvalidCityStateZipInvalidZipFormat() {
69         assertFalse(regexValidator.validLocation(theLocation:"Seattle, WA 9810")); //Test Invalid zip format
70     }
71
72     @Test
73     public void testInvalidCityStateZipSpecialCharacters() {
74         assertFalse(regexValidator.validLocation(theLocation:"Seattle! WA 98101")); //Test Special characters in city
75     }
76
77     @Test
78     public void testInvalidCityStateZipNumericCity() {
79         assertFalse(regexValidator.validLocation(theLocation:"123, WA 98101")); //Test Numeric city
80     }
81
82     @Test
83     public void testInvalidCityStateZipEmptyString() {
84         assertFalse(regexValidator.validLocation(theLocation:"")); //Test Empty string
85     }
86 }
87

```

## 8. Military Time:

```
1  import org.junit.jupiter.api.Test;
2  import static org.junit.jupiter.api.Assertions.*;
3
4  public class MilitaryTimeValidationTest {
5      @Test
6      public void testValidMilitaryTimeMidnight() { // Test valid military time for midnight (0000)
7          assertTrue(regexValidator.validMilitaryTime(theTime:"0000"));
8      }
9
10     @Test
11     public void testValidMilitaryTimeNoon() { // Test valid military time for noon (1200)
12         assertTrue(regexValidator.validMilitaryTime(theTime:"1200"));
13     }
14
15     @Test
16     public void testValidMilitaryTimeEarlyMorning() { // Test valid military time for early morning (0530)
17         assertTrue(regexValidator.validMilitaryTime(theTime:"0530"));
18     }
19
20     @Test
21     public void testValidMilitaryTimeSingleDigitHour() { // Test valid military time with a single-digit hour (0900)
22         assertTrue(regexValidator.validMilitaryTime(theTime:"0900"));
23     }
24
25     @Test
26     public void testValidMilitaryTimeFullDay() { // Test valid military time for the last minute of the day (2359)
27         assertTrue(regexValidator.validMilitaryTime(theTime:"2359"));
28     }
29
30     @Test
31     public void testValidMilitaryTimeEveningTime() { // Test valid military time for evening time (1845)
32         assertTrue(regexValidator.validMilitaryTime(theTime:"1845"));
33     }
34
35     @Test
36     public void testValidMilitaryTimeLeadingZero() { // Test valid military time with a leading zero (0005)
37         assertTrue(regexValidator.validMilitaryTime(theTime:"0005"));
38     }
39
40     @Test
41     public void testValidMilitaryTimeSingleDigitMinute() { // Test valid military time with a single-digit minute (1205)
42         assertTrue(regexValidator.validMilitaryTime(theTime:"1205"));
43     }
44 }
```

```

45     @Test
46     public void testInvalidMilitaryTimeInvalidHour() { // Test invalid military time with an invalid hour (2460)
47         assertFalse(regexValidator.validMilitaryTime(theTime:"2460"));
48     }
49
50     @Test
51     public void testInvalidMilitaryTimeInvalidMinute() { // Test invalid military time with an invalid minute (1260)
52         assertFalse(regexValidator.validMilitaryTime(theTime:"1260"));
53     }
54
55     @Test
56     public void testInvalidMilitaryTimeNonNumeric() { // Test invalid military time with non-numeric characters (12AB)
57         assertFalse(regexValidator.validMilitaryTime(theTime:"12AB"));
58     }
59
60     @Test
61     public void testInvalidMilitaryTimeTooShort() { // Test invalid military time that is too short (123)
62         assertFalse(regexValidator.validMilitaryTime(theTime:"123"));
63     }
64
65     @Test
66     public void testInvalidMilitaryTimeTooLong() { // Test invalid military time that is too long (123456)
67         assertFalse(regexValidator.validMilitaryTime(theTime:"123456"));
68     }
69
70     @Test
71     public void testInvalidMilitaryTimeWithColon() { // Test invalid military time with a colon (12:30)
72         assertFalse(regexValidator.validMilitaryTime(theTime:"12:30"));
73     }
74
75     @Test
76     public void testInvalidMilitaryTimeEmpty() { // Test invalid military time with an empty string
77         assertFalse(regexValidator.validMilitaryTime(theTime:""));
78     }
79
80     @Test
81     public void testInvalidMilitaryTimeNegative() { // Test invalid military time with a negative sign (-1200)
82         assertFalse(regexValidator.validMilitaryTime(theTime:"-1200"));
83     }
84 }
85

```

## 9. US Currency:

```

1  import org.junit.jupiter.api.Test;
2  import static org.junit.jupiter.api.Assertions.*;
3
4  public class CurrencyValidationTest {
5
6      // Test valid currency formats
7      @Test
8      public void testValidCurrencySimpleValue() { //Tests a simple currency value with two decimal numbers
9          assertTrue(regexValidator.validUSCurrency(theCurrency: "$100.00"));
10     }
11
12     @Test
13     public void testValidCurrencyWithCommas() { //Tests a large currency value with commas
14         assertTrue(regexValidator.validUSCurrency(theCurrency: "$1,234,567.89"));
15     }
16
17     @Test
18     public void testValidCurrencyWholeNumber() { //Tests a whole number without decimal places
19         assertTrue(regexValidator.validUSCurrency(theCurrency: "$500"));
20     }
21
22     @Test
23     public void testValidCurrencyNegativeValue() { //Tests a negative currency value
24         assertTrue(regexValidator.validUSCurrency(theCurrency: "-$100.00"));
25     }
26
27     @Test
28     public void testValidCurrencyLargeNumber() { //Tests a large currency value with commas and decimal places
29         assertTrue(regexValidator.validUSCurrency(theCurrency: "$123,456,789.00"));
30     }
31
32     @Test
33     public void testValidCurrencyCentsOnly() { //Tests a currency value with only cents
34         assertTrue(regexValidator.validUSCurrency(theCurrency: "$0.99"));
35     }
36
37     @Test
38     public void testValidCurrencyZero() { //Tests a zero currency value
39         assertTrue(regexValidator.validUSCurrency(theCurrency: "$0.00"));
40     }
41
42     @Test
43     public void testValidCurrencyWithoutCents() { //Tests a currency value without cents
44         assertTrue(regexValidator.validUSCurrency(theCurrency: "$1,000"));
45     }
46

```



```

46
47 // Test invalid currency formats
48 @Test
49 public void testInvalidCurrencyMissingDollarSign() { //Tests a missing dollar sign
50     assertFalse(regexValidator.validUSCurrency(theCurrency:"100.00"));
51 }
52
53 @Test
54 public void testInvalidCurrencyInvalidFormat() { //Tests an incorrectly formatted currency value
55     assertFalse(regexValidator.validUSCurrency(theCurrency:"$100,00"));
56 }
57
58 @Test
59 public void testInvalidCurrencyTooManyCents() { //Tests a currency value with more than two decimal places
60     assertFalse(regexValidator.validUSCurrency(theCurrency:"$100.001"));
61 }
62
63 @Test
64 public void testInvalidCurrencyNonNumericChars() { //Tests a currency value containing non-numeric characters
65     assertFalse(regexValidator.validUSCurrency(theCurrency:"$10A.00"));
66 }
67
68 @Test
69 public void testInvalidCurrencyMultipleDollarSigns() { //Tests a currency value with multiple dollar signs
70     assertFalse(regexValidator.validUSCurrency(theCurrency:"$$100.00"));
71 }
72
73 @Test
74 public void testInvalidCurrencyInvalidCommaPlacement() { //Tests an invalid comma placement in the currency value
75     assertFalse(regexValidator.validUSCurrency(theCurrency:"$1,00.00"));
76 }
77
78 @Test
79 public void testInvalidCurrencyEmptyString() { //Tests an empty string
80     assertFalse(regexValidator.validUSCurrency(theCurrency:""));
81 }
82
83 @Test
84 public void testInvalidCurrencySpaces() { //Tests a currency value with spaces after the dollar sign
85     assertFalse(regexValidator.validUSCurrency(theCurrency:"$ 100.00"));
86 }
87 }
88

```

## 10. URLs:

```

1  import org.junit.jupiter.api.Test;
2  import static org.junit.jupiter.api.Assertions.*;
3
4  public class URLValidationTest {
5      @Test
6      public void testValidURLHttps() { // Test valid URL with HTTPS protocol
7          assertTrue(regexValidator.isValidURL(theLinks:"https://www.example.com"));
8      }
9
10     @Test
11     public void testValidURLHttp() { // Test valid URL with HTTP protocol
12         assertTrue(regexValidator.isValidURL(theLinks:"http://example.com"));
13     }
14
15     @Test
16     public void testValidURLNoProtocol() { // Test valid URL without any protocol
17         assertTrue(regexValidator.isValidURL(theLinks:"www.example.com"));
18     }
19
20     @Test
21     public void testValidURLWithSubdomain() { // Test valid URL with subdomain
22         assertTrue(regexValidator.isValidURL(theLinks:"https://subdomain.example.co.uk"));
23     }
24
25     @Test
26     public void testValidURLWithPort() { // Test valid URL with a port number
27         assertTrue(regexValidator.isValidURL(theLinks:"http://localhost:8080"));
28     }
29
30     @Test
31     public void testValidURLWithPath() { // Test valid URL with a path
32         assertTrue(regexValidator.isValidURL(theLinks:"https://example.com/path/to/page"));
33     }
34
35     @Test
36     public void testValidURLWithQueryParams() { // Test valid URL with query parameters
37         assertTrue(regexValidator.isValidURL(theLinks:"https://example.com/search?q=test&page=1"));
38     }
39
40     @Test
41     public void testValidURLMixedCase() { // Test valid URL with mixed case in the domain
42         assertTrue(regexValidator.isValidURL(theLinks:"HTTPS://Example.COM"));
43     }
44

```

```

45 // Invalid URL Tests
46 @Test
47 public void testInvalidURLMissingDomain() { // Test invalid URL with missing domain
48     assertFalse(regexValidator.isValidURL(theLinks:"https://"));
49 }
50
51 @Test
52 public void testInvalidURLInvalidProtocol() { // Test invalid URL with an unsupported protocol
53     assertFalse(regexValidator.isValidURL(theLinks:"ftp://example.com"));
54 }
55
56 @Test
57 public void testInvalidURLMissingDot() { // Test invalid URL missing a dot in the domain name
58     assertFalse(regexValidator.isValidURL(theLinks:"https://examplecom"));
59 }
60
61 @Test
62 public void testInvalidURLInvalidChars() { // Test invalid URL with invalid characters in the domain
63     assertFalse(regexValidator.isValidURL(theLinks:"https://example!.com"));
64 }
65
66 @Test
67 public void testInvalidURLSpaces() { // Test invalid URL with spaces in the domain
68     assertFalse(regexValidator.isValidURL(theLinks:"https://example domain.com"));
69 }
70
71 @Test
72 public void testInvalidURLEmptyString() { // Test invalid URL with an empty string
73     assertFalse(regexValidator.isValidURL(theLinks:""));
74 }
75
76 @Test
77 public void testInvalidURLOnlyProtocol() { // Test invalid URL with only the protocol and no domain
78     assertFalse(regexValidator.isValidURL(theLinks:"https://"));
79 }
80
81 @Test
82 public void testInvalidURLInvalidTopLevelDomain() { // Test invalid URL with an invalid top-level domain
83     assertFalse(regexValidator.isValidURL(theLinks:"https://example.invalidtld"));
84 }
85 }
86

```

11. Password:

```

1  import org.junit.jupiter.api.Test;
2  import static org.junit.jupiter.api.Assertions.*;
3
4  public class PassWordValidationTest {
5
6      @Test
7      public void testValidPasswordMinimumLength() {
8          assertTrue(regexValidator.validPassword(thePassword:"Abc!123456")); // Exactly 10 characters, meets all conditions
9      }
10
11      @Test
12      public void testValidPasswordExceedsMinimumLength() {
13          assertTrue(regexValidator.validPassword(thePassword:"XyZ!45678abc")); // More than 10 characters, still valid
14      }
15
16      @Test
17      public void testValidPasswordWithMaxConsecutiveLowercase() {
18          assertTrue(regexValidator.validPassword(thePassword:"Aabc!123DEF")); // 3 consecutive lowercase allowed
19      }
20
21      @Test
22      public void testValidPasswordWithMixedCases() {
23          assertTrue(regexValidator.validPassword(thePassword:"Pass1!WordX")); // Proper mix of upper, lower, digit, punctu
24      }
25
26      @Test
27      public void testValidPasswordWithSinglePunctuation() {
28          assertTrue(regexValidator.validPassword(thePassword:"Secure!2Pass12")); // Uses exactly one punctuation mark
29      }
30
31      @Test
32      public void testValidPasswordWithMultipleDigits() {
33          assertTrue(regexValidator.validPassword(thePassword:"Strong!9876Abc")); // Extra numbers but still valid
34      }
35
36      @Test
37      public void testValidPasswordWithDifferentPunctuation() {
38          assertTrue(regexValidator.validPassword(thePassword:"Hello!123World")); // '@' as punctuation
39      }
40
41      @Test
42      public void testValidPasswordWithDifferentPunctuation2() {
43          assertTrue(regexValidator.validPassword(thePassword:"XyZ!45678abc")); // '#' as punctuation
44      }
45

```

```
46     @Test
47     public void testInvalidPasswordTooShort() {
48         assertFalse(regexValidator.validatePassword(thePassword:"Ab1!cde")); // Less than 10 characters
49     }
50
51     @Test
52     public void testInvalidPasswordMissingUppercase() {
53         assertFalse(regexValidator.validatePassword(thePassword:"abcdef!1234")); // No uppercase Letter
54     }
55
56     @Test
57     public void testInvalidPasswordMissingLowercase() {
58         assertFalse(regexValidator.validatePassword(thePassword:"ABCDEFGH!1234")); // No lowercase Letter
59     }
60
61     @Test
62     public void testInvalidPasswordMissingDigit() {
63         assertFalse(regexValidator.validatePassword(thePassword:"Abcdefgh!X")); // No digit
64     }
65
66     @Test
67     public void testInvalidPasswordMissingPunctuation() {
68         assertFalse(regexValidator.validatePassword(thePassword:"Abcdef12345")); // No punctuation
69     }
70
71     @Test
72     public void testInvalidPasswordTooManyConsecutiveLowercase() {
73         assertFalse(regexValidator.validatePassword(thePassword:"Abcdeeee!123")); // More than 3 consecutive Lowercase Letter
74     }
75
76     @Test
77     public void testInvalidPasswordTooManyConsecutiveLowercase2() {
78         assertFalse(regexValidator.validatePassword(thePassword:"Abcdaaaa!123")); // More than 3 consecutive Lowercase Letter
79     }
80
81     @Test
82     public void testInvalidPasswordWithMultiplePunctuation() {
83         assertFalse(regexValidator.validatePassword(thePassword:"Secure!Pass@12")); // More than one punctuation mark
84     }
85
```

```

86     @Test
87     public void testInvalidPasswordWithOnlyLettersNoDigitOrPunctuation() {
88         assertFalse(regexValidator.validatePassword(thePassword:"Abcdefghijkl")); // No digit, no punctuation
89     }
90
91     @Test
92     public void testInvalidPasswordWithOnlyDigitsAndNoLetters() {
93         assertFalse(regexValidator.validatePassword(thePassword:"1234567890!")); // No upper/Lowercase Letters
94     }
95
96     @Test
97     public void testInvalidPasswordWithOnlyPunctuationAndLetters() {
98         assertFalse(regexValidator.validatePassword(thePassword:"Abcdef!!!")); // No digit
99     }
100
101     @Test
102     public void testInvalidPasswordWithSpaces() {
103         assertFalse(regexValidator.validatePassword(thePassword:"Abc 123!XYZ")); // Spaces should not be allowed
104     }
105
106     @Test
107     public void testInvalidPasswordWithTabs() {
108         assertFalse(regexValidator.validatePassword(thePassword:"Abc\t123!XYZ")); // Tabs should not be allowed
109     }
110
111     @Test
112     public void testInvalidPasswordWithNewlineCharacters() {
113         assertFalse(regexValidator.validatePassword(thePassword:"Abc\n123!XYZ")); // NewLines should not be allowed
114     }
115
116     @Test
117     public void testInvalidPasswordWithTooManyConsecutiveRepeatingCharacters() {
118         assertFalse(regexValidator.validatePassword(thePassword:"Aaaab!123X")); // More than 3 consecutive 'a' characters
119     }
120
121     @Test
122     public void testInvalidPasswordWithOnlyUppercaseLetters() {
123         assertFalse(regexValidator.validatePassword(thePassword:"HELL0123!X")); // No Lowercase Letter
124     }
125 }
126

```

12. All words containing an odd number of alphabetic characters, ending in "ion".

```
1  import static org.junit.Assert.assertFalse;
2  import static org.junit.Assert.assertTrue;
3
4  import org.junit.Test;
5
6  public class OddWordValidationTest {
7
8      @Test
9      public void testValidOddWordShortest() {
10         assertTrue(regexValidator.validlengthWord(word:"ion")); // 3 total Letters (odd)
11     }
12
13     @Test
14     public void testValidOddWordFiveLetters() {
15         assertTrue(regexValidator.validlengthWord(word:"union")); // 5 total Letters (odd)
16     }
17
18     @Test
19     public void testValidOddWordSevenLetters() {
20         assertTrue(regexValidator.validlengthWord(word:"invention")); // 7 total Letters (odd)
21     }
22
23     @Test
24     public void testValidOddWordMixedCase() {
25         assertTrue(regexValidator.validlengthWord(word:"Suppression")); // Case insensitive check
26     }
27
28     @Test
29     public void testValidOddWordThirteenLetters() {
30         assertTrue(regexValidator.validlengthWord(word:"participation")); // 13 total Letters (odd)
31     }
32
33     @Test
34     public void testValidOddWordNineteenLetters() {
35         assertTrue(regexValidator.validlengthWord(word:"conceptualization")); // 19 total Letters (odd)
36     }
37
38     @Test
39     public void testValidOddWordTwentyThreeLetters() {
40         assertTrue(regexValidator.validlengthWord(word:"Overintellectualization")); // 23 total Letters (odd)
41     }
42
43     @Test
44     public void testValidOddWordWithCapitalization() {
45         assertTrue(regexValidator.validlengthWord(word:"Commercialization")); // Should be case insensitive
46     }
47 }
```

```

48     @Test
49     public void testInvalidWordWrongEnding() {
50         assertFalse(regexValidator.validlengthWord(word:"motivation")); // not correct Length (even)
51     }
52
53     @Test
54     public void testInvalidWordWithNumbers() {
55         assertFalse(regexValidator.validlengthWord(word:"m0tion")); // Contains a number
56     }
57
58     @Test
59     public void testInvalidWordWithSpecialCharacters() {
60         assertFalse(regexValidator.validlengthWord(word:"moti@nion")); // Contains special character
61     }
62
63     @Test
64     public void testInvalidWordWithSpaces() {
65         assertFalse(regexValidator.validlengthWord(word:"expulsion ")); // Leading/trailing spaces
66     }
67
68     @Test
69     public void testInvalidWordWithIncorrectPattern() {
70         assertFalse(regexValidator.validlengthWord(word:"bahion")); // Doesn't follow the regex pattern
71     }
72
73     @Test
74     public void testInvalidWordEvenLengthFourLetters() {
75         assertFalse(regexValidator.validlengthWord(word:"xion")); // 4 Letters (even)
76     }
77
78     @Test
79     public void testInvalidWordEvenLengthFourteenLetters() {
80         assertFalse(regexValidator.validlengthWord(word:"disintegration")); // 14 total Letters (even)
81     }
82
83     @Test
84     public void testInvalidWordNoIonEnding() {
85         assertFalse(regexValidator.validlengthWord(word:"apple")); // Doesn't end in "ion"
86     }
87 }
88

```

All test passed:

225/225
 2.3s

- RegularExpressions 150ms
  - {} <Default Package> 150ms
    - > AddressValidationTest 8.0ms
    - > CurrencyValidationTest 8.0ms
    - > DateValidationTest 10ms
    - > EmailValidationTest 8.0ms
    - > ExtraCreditTest 12ms
    - > LocationValidationTest 19ms
    - > MilitaryTimeValidationTest 24ms
    - > NameValidationTest 8.0ms
    - > OddWordValidationTest 8.0ms
    - > PassWordValidationTest 10ms
    - > PhoneNumberValidationTest 1...
    - > SSNValidationTest 13ms
    - > URLValidationTest 12ms



Extra Credit for all 3:

```
1  import org.junit.Test;
2  import static org.junit.Assert.*;
3
4  public class ExtraCreditTest {
5      // SSN Extra Credit Tests - Testing SSA numbering rules
6      @Test
7      public void testInvalidSSNArea666() {
8          assertFalse(regexValidator.validSSN(theSSN:"666-45-6789"));
9      }
10
11      @Test
12      public void testInvalidSSNArea000() {
13          assertFalse(regexValidator.validSSN(theSSN:"000-45-6789"));
14      }
15
16      @Test
17      public void testInvalidSSNArea900() {
18          assertFalse(regexValidator.validSSN(theSSN:"900-45-6789"));
19      }
20
21      @Test
22      public void testInvalidSSNArea999() {
23          assertFalse(regexValidator.validSSN(theSSN:"999-45-6789"));
24      }
25
26      @Test
27      public void testValidSSNAreaNumber() {
28          assertTrue(regexValidator.validSSN(theSSN:"123-45-6789"));
29      }
30
31      @Test
32      public void testInvalidSSNGroup00() {
33          assertFalse(regexValidator.validSSN(theSSN:"123-00-6789"));
34      }
35
36      @Test
37      public void testInvalidSSNSerial0000() {
38          assertFalse(regexValidator.validSSN(theSSN:"123-45-0000"));
39      }
40
41      @Test
42      public void testValidSSNBoundaryArea() {
43          assertTrue(regexValidator.validSSN(theSSN:"899-45-6789"));
44      }
45
46      // ... (Additional tests for SSN format) ...
47  }
```

```
46 // Phone Number Extra Credit Tests - Testing official area codes
47 @Test
48 public void testValidAreaCode212() {
49     assertTrue(regexValidator.validPhoneNumber(thePhoneNumber: "(212)555-1234")); // NYC
50 }
51
52 @Test
53 public void testValidAreaCode415() {
54     assertTrue(regexValidator.validPhoneNumber(thePhoneNumber: "(415)555-1234")); // San Francisco
55 }
56
57 @Test
58 public void testValidAreaCode305() {
59     assertTrue(regexValidator.validPhoneNumber(thePhoneNumber: "(305)555-1234")); // Miami
60 }
61
62 @Test
63 public void testInvalidAreaCode000() {
64     assertFalse(regexValidator.validPhoneNumber(thePhoneNumber: "(000)555-1234"));
65 }
66
67 @Test
68 public void testInvalidAreaCode001() {
69     assertFalse(regexValidator.validPhoneNumber(thePhoneNumber: "(001)555-1234"));
70 }
71
72 @Test
73 public void testInvalidAreaCode123() {
74     assertFalse(regexValidator.validPhoneNumber(thePhoneNumber: "(123)555-1234")); // Invalid area code
75 }
76
77 @Test
78 public void testInvalidAreaCode999() {
79     assertFalse(regexValidator.validPhoneNumber(thePhoneNumber: "(999)555-1234"));
80 }
81
82 @Test
83 public void testInvalidAreaCodeAlpha() {
84     assertFalse(regexValidator.validPhoneNumber(thePhoneNumber: "(ABC)555-1234"));
85 }
86
```

```

87 // State Abbreviation Extra Credit Tests
88 @Test
89 public void testValidStateWA() {
90     assertTrue(regexValidator.validLocation(theLocation:"Seattle, WA 98101"));
91 }
92
93 @Test
94 public void testValidStateNY() {
95     assertTrue(regexValidator.validLocation(theLocation:"New York, NY 10001"));
96 }
97
98 @Test
99 public void testValidStateCA() {
100     assertTrue(regexValidator.validLocation(theLocation:"Los Angeles, CA 90001"));
101 }
102
103 @Test
104 public void testInvalidStateXX() {
105     assertFalse(regexValidator.validLocation(theLocation:"Invalid, XX 12345"));
106 }
107
108 @Test
109 public void testInvalidState00() {
110     assertFalse(regexValidator.validLocation(theLocation:"Invalid, 00 12345"));
111 }
112
113 @Test
114 public void testInvalidStateLowercase() {
115     assertFalse(regexValidator.validLocation(theLocation:"Invalid, wa 12345"));
116 }
117
118 @Test
119 public void testInvalidStateMixedCase() {
120     assertFalse(regexValidator.validLocation(theLocation:"Invalid, Wa 12345"));
121 }
122
123 @Test
124 public void testInvalidStateThreeLetters() {
125     assertFalse(regexValidator.validLocation(theLocation:"Invalid, WAA 12345"));
126 }
127 }

```