Individual Assignment - Regular Expressions JUnit Test

Name: Sopheanith Ny

---

1. SSN:

```java
import org.junit.jupiter.api.Test;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

public class SSNValidationTest {

    @Test
    void testValidSSNAllDigits() { // Test valid SSN with all digits (123456789)
        assertTrue(regexValidator.validSSN(theSSN:"123456789"));
    }

    @Test
    void testValidSSNDashes() { // Test valid SSN with dashes (123-45-6789)
        assertTrue(regexValidator.validSSN(theSSN:"123-45-6789"));
    }

    @Test
    void testValidSSNSpaces() { // Test valid SSN with spaces (123 45 6789)
        assertTrue(regexValidator.validSSN(theSSN:"123 45 6789"));
    }

    @Test
    void testValidSSNWithEdgeCaseLowest() { // Test valid SSN with edge case (lowest possible SSN 001-01-0001)
        assertTrue(regexValidator.validSSN(theSSN:"001-01-0001"));
    }

    @Test
    void testValidSSNWithEdgeCaseHighest() { // Test valid SSN with edge case (highest possible SSN 899-99-9999)
        assertTrue(regexValidator.validSSN(theSSN:"899-99-9999"));
    }

    @Test
    void testValidSSNWithMidRange() { // Test valid SSN with mid-range numbers (456-78-9012)
        assertTrue(regexValidator.validSSN(theSSN:"456-78-9012"));
    }

    @Test
    void testValidSSNWithDifferentNumbers() { // Test valid SSN with different numbers (212-58-4567)
        assertTrue(regexValidator.validSSN(theSSN:"212-58-4567"));
    }

    @Test
    void testValidSSNWithSpecialCase() { // Test valid SSN with a special case (078-05-1120)
        assertTrue(regexValidator.validSSN(theSSN:"078-05-1120"));
    }

```

```
    @Test
    void testInvalidSSNAllZeroes() { // Test invalid SSN with all zeroes (000-00-0000)
        assertFalse(regexValidator.validSSN(theSSN:"000-00-0000"));
    }

    @Test
    void testInvalidSSNStartsWith666() { // Test invalid SSN starting with 666 (666-00-1234)
        assertFalse(regexValidator.validSSN(theSSN:"666-00-1234"));
    }

    @Test
    void testInvalidSSNWithLetters() { // Test invalid SSN with letters (abc-de-ghij)
        assertFalse(regexValidator.validSSN(theSSN:"abc-de-ghij"));
    }

    @Test
    void testInvalidSSNTooManyDigits() { // Test invalid SSN with too many digits (999-99-99999)
        assertFalse(regexValidator.validSSN(theSSN:"999-99-99999"));
    }

    @Test
    void testInvalidSSNTooFewDigits() { // Test invalid SSN with too few digits (123-45-67)
        assertFalse(regexValidator.validSSN(theSSN:"123-45-67"));
    }

    @Test
    void testInvalidSSNWrongSeparators() { // Test invalid SSN with wrong separators (123.45.6789)
        assertFalse(regexValidator.validSSN(theSSN:"123.45.6789"));
    }

    @Test
    void testInvalidSSNWithInvalidAreaNumber() { // Test invalid SSN with invalid area number (900-12-3456)
        assertFalse(regexValidator.validSSN(theSSN:"900-12-3456"));
    }

    @Test
    void testInvalidSSNEmptyString() { // Test invalid SSN with empty string
        assertFalse(regexValidator.validSSN(theSSN:""));
    }

    // Extra credit tests for SSN numbering rules
    @Test
    public void testInvalidSSNZeroGroup() { // Test invalid SSN with zero group (000-00-0000)
        assertFalse(regexValidator.validSSN(theSSN:"000-00-0000"));
    }

    @Test
    public void testInvalidSSNInvalidPrefix() { // Test invalid SSN with invalid prefix (666-12-3456)
        assertFalse(regexValidator.validSSN(theSSN:"666-12-3456"));
    }
}
```

2. US Phone Number:

```java
import org.junit.jupiter.api.Test;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

public class PhoneNumberValidationTest {

    @Test
    void testValidPhoneNumberWithDashes() { // Test valid phone number with dashes (213-456-7890)
        assertTrue(regexValidator.validPhoneNumber(thePhoneNumber:"213-456-7890"));
    }

    @Test
    void testValidPhoneNumberWithDots() { // Test valid phone number with dots (415.456.7890)
        assertTrue(regexValidator.validPhoneNumber(thePhoneNumber:"415.456.7890"));
    }

    @Test
    void testValidPhoneNumberWithSpaces() { // Test valid phone number with spaces (718 456 7890)
        assertTrue(regexValidator.validPhoneNumber(thePhoneNumber:"718 456 7890"));
    }

    @Test
    void testValidPhoneNumberWithParentheses() { // Test valid phone number with parentheses (503) 456-7890
        assertTrue(regexValidator.validPhoneNumber(thePhoneNumber:"(503) 456-7890"));
    }

    @Test
    void testValidPhoneNumberWithCountryCode() { // Test valid phone number with country code (+1 646-456-7890)
        assertTrue(regexValidator.validPhoneNumber(thePhoneNumber:"+1 646-456-7890"));
    }

    @Test
    void testValidPhoneNumberWithoutSeparators() { // Test valid phone number without separators (8184567890)
        assertTrue(regexValidator.validPhoneNumber(thePhoneNumber:"8184567890"));
    }

    @Test
    void testValidPhoneNumberWithDifferentAreaCode() { // Test valid phone number with different area code (707)999-9999
        assertTrue(regexValidator.validPhoneNumber(thePhoneNumber:"(707)999-9999"));
    }

    @Test
    void testValidPhoneNumberWithMidRange() { // Test valid phone number with mid-range area code (212) 555-7890
        assertTrue(regexValidator.validPhoneNumber(thePhoneNumber:"(212) 555-7890"));
    }

```

```java
    @Test
    void testInvalidPhoneNumberTooShort() { // Test invalid phone number that is too short (999-9999)
        assertFalse(regexValidator.validPhoneNumber(thePhoneNumber:"999-9999"));
    }

    @Test
    void testInvalidPhoneNumberWithLetters() { // Test invalid phone number with letters (abcdefghij)
        assertFalse(regexValidator.validPhoneNumber(thePhoneNumber:"abcdefghij"));
    }

    @Test
    void testInvalidPhoneNumberWrongSeparator() { // Test invalid phone number with wrong separator (123/456/7890)
        assertFalse(regexValidator.validPhoneNumber(thePhoneNumber:"123/456/7890"));
    }

    @Test
    void testInvalidPhoneNumberExtraDigit() { // Test invalid phone number with extra digit (123-456-78901)
        assertFalse(regexValidator.validPhoneNumber(thePhoneNumber:"123-456-78901"));
    }

    @Test
    void testInvalidPhoneNumberWithMissingDigit() { // Test invalid phone number with missing digit (123-456-789)
        assertFalse(regexValidator.validPhoneNumber(thePhoneNumber:"123-456-789"));
    }

    @Test
    void testInvalidPhoneNumberWithExtraCharacters() { // Test invalid phone number with extra characters ((123) 456-7890
        assertFalse(regexValidator.validPhoneNumber(thePhoneNumber:"(123) 456-7890 x123"));
    }

    @Test
    void testInvalidPhoneNumberWithInvalidAreaCode() { // Test invalid phone number with invalid area code (000) 456-7890
        assertFalse(regexValidator.validPhoneNumber(thePhoneNumber:"(000) 456-7890"));
    }

    @Test
    void testInvalidPhoneNumberEmptyString() { // Test invalid phone number with an empty string
        assertFalse(regexValidator.validPhoneNumber(thePhoneNumber:""));
    }
}
```

3. Email Address:

```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class EmailValidationTest {

    @Test
    void testValidEmailSimple() { //Tests a simple email format
        assertTrue(regexValidator.validEmail(theEmail:"john.doe@example.com"));
    }

    @Test
    void testValidEmailWithNumbers() { //Tests an email containing numbers in the local part
        assertTrue(regexValidator.validEmail(theEmail:"user123@domain.net"));
    }

    @Test
    void testValidEmailWithUnderscore() { //Tests an email containing an underscore
        assertTrue(regexValidator.validEmail(theEmail:"user_name@example.org"));
    }

    @Test
    void testValidEmailWithDash() { //Tests an email containing a hyphen in the local part
        assertTrue(regexValidator.validEmail(theEmail:"user-name@domain.com"));
    }

    @Test
    void testValidEmailWithSubdomain() { //Tests an email with a subdomain
        assertTrue(regexValidator.validEmail(theEmail:"contact@sub.example.co.uk"));
    }

    @Test
    void testValidEmailWithLongTLD() { //Tests an email with a long top-level domain (TLD)
        assertTrue(regexValidator.validEmail(theEmail:"person@company.travel"));
    }

    @Test
    void testValidEmailWithCapitalLetters() { //Tests an email with capital letters
        assertTrue(regexValidator.validEmail(theEmail:"John.Doe@Example.COM"));
    }

    @Test
    void testValidEmailWithNumbersAndHyphens() { //Tests an email with numbers and hyphens
        assertTrue(regexValidator.validEmail(theEmail:"test-email123@company.biz"));
    }

```

```java
    @Test
    void testInvalidEmailMissingAtSymbol() { //Tests an invalid email missing the '@' symbol
        assertFalse(regexValidator.validEmail(theEmail:"userexample.com"));
    }

    @Test
    void testInvalidEmailMultipleAtSymbols() { //Tests an invalid email with multiple '@' symbols
        assertFalse(regexValidator.validEmail(theEmail:"user@@example.com"));
    }

    @Test
    void testInvalidEmailStartingWithDot() { //Tests an invalid email starting with a dot
        assertFalse(regexValidator.validEmail(theEmail:".user@example.com"));
    }

    @Test
    void testInvalidEmailEndingWithDot() { //Tests an invalid email ending with a dot before @
        assertFalse(regexValidator.validEmail(theEmail:"user.@example.com"));
    }

    @Test
    void testInvalidEmailNoDomain() { //Tests an invalid email with no domain name
        assertFalse(regexValidator.validEmail(theEmail:"user@.com"));
    }

    @Test
    void testInvalidEmailInvalidCharacters() { //Tests an invalid email containing special characters
        assertFalse(regexValidator.validEmail(theEmail:"user!name@example.com"));
    }

    @Test
    void testInvalidEmailDoubleDots() { //Tests an invalid email with consecutive dots in the local part
        assertFalse(regexValidator.validEmail(theEmail:"user..name@example.com"));
    }

    @Test
    void testInvalidEmailNoTLD() { //Tests an invalid email missing a domain
        assertFalse(regexValidator.validEmail(theEmail:"user@example"));
    }
}

```

4. Full Name:

```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class NameValidationTest {
    @Test
    void testValidNameWithoutMiddleInitial() { // Test valid name without middle initial (Smith, John)
        assertTrue(regexValidator.validName(theName:"Smith, John"));
    }

    @Test
    void testValidNameWithMiddleInitial() { // Test valid name with middle initial (Smith, John L)
        assertTrue(regexValidator.validName(theName:"Smith, John L"));
    }

    @Test
    void testValidNameWithDoubleLastName() { // Test valid name with double last name (Smith-Jones, Mary K)
        assertTrue(regexValidator.validName(theName:"Smith-Jones, Mary K"));
    }

    @Test
    void testValidNameWithApostrophe() { // Test valid name with apostrophe in last name (O'Connor, Liam P)
        assertTrue(regexValidator.validName(theName:"O'Connor, Liam P"));
    }

    @Test
    void testValidNameWithMultipleMiddleInitials() { // Test valid name with multiple middle initials (Brown, Sarah J K)
        assertTrue(regexValidator.validName(theName:"Brown, Sarah J K"));
    }

    @Test
    void testValidNameWithHyphenatedFirstName() { // Test valid name with hyphenated first name (Lee, Anne-Marie)
        assertTrue(regexValidator.validName(theName:"Lee, Anne-Marie"));
    }

    @Test
    void testValidNameWithMiddleInitialLowerCase() { // Test invalid name with lowercase middle initial (Smith, John l)
        assertFalse(regexValidator.validName(theName:"Smith, John l"));
    }

    @Test
    void testValidNameWithExtraSpaces() { // Test invalid name with extra spaces around name (  Smith, John  )
        assertFalse(regexValidator.validName(theName:"  Smith, John  "));
    }

    @Test
    void testInvalidNameMissingComma() { // Test invalid name missing a comma (Smith John)
        assertFalse(regexValidator.validName(theName:"Smith John"));
    }

    @Test
    void testInvalidNameWithNumbers() { // Test invalid name containing numbers (Johnson, Mark 2)
        assertFalse(regexValidator.validName(theName:"Johnson, Mark 2"));
    }

    @Test
    void testInvalidNameWithSpecialCharacters() { // Test invalid name with special characters (Doe, @lice)
        assertFalse(regexValidator.validName(theName:"Doe, @lice"));
    }

    @Test
    void testInvalidNameOnlyComma() { // Test invalid name with only a comma (,)
        assertFalse(regexValidator.validName(theName:","));
    }

    @Test
    void testInvalidNameMiddleInitialTooLong() { // Test invalid name with a middle initial too long (Davis, Chris XYZ)
        assertFalse(regexValidator.validName(theName:"Davis, Chris XYZ"));
    }

    @Test
    void testInvalidNameOnlyFirstName() { // Test invalid name with only a first name (John)
        assertFalse(regexValidator.validName(theName:"John"));
    }

    @Test
    void testInvalidNameOnlyLastName() { // Test invalid name with only a last name (Smith,)
        assertFalse(regexValidator.validName(theName:"Smith,"));
    }
}
```

5. Date in MM-DD-YYYY:

```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class DateValidationTest {

    // Test valid date formats
    @Test
    public void testValidDateStandardFormat() { //Tests a standard date format with dashes
        assertTrue(regexValidator.validDate(theDate:"12-31-2023"));
    }

    @Test
    public void testValidDateSlashSeparator() { //Tests a standard date format with slashes
        assertTrue(regexValidator.validDate(theDate:"12/31/2023"));
    }

    @Test
    public void testValidLeapYearDate() { //Tests a leap year date
        assertTrue(regexValidator.validDate(theDate:"02-29-2024"));
    }

    @Test
    public void testValidEdgeCaseFirstDayOfYear() { //Tests the first day of the year
        assertTrue(regexValidator.validDate(theDate:"01-01-2023"));
    }

    @Test
    public void testValidEdgeCaseLastDayOfYear() { //Tests the last day of the year
        assertTrue(regexValidator.validDate(theDate:"12-31-2023"));
    }

    @Test
    public void testValidShortYearFormat() { //Tests a short year format
        assertTrue(regexValidator.validDate(theDate:"12-31-23"));
    }

    @Test
    public void testValidMonthWithLeadingZero() { //Tests a month with a leading zero
        assertTrue(regexValidator.validDate(theDate:"02-15-2024"));
    }

    @Test
    public void testValidMonthWithoutLeadingZero() { //Tests a month without a leading zero
        assertTrue(regexValidator.validDate(theDate:"2-15-2024"));
    }

```

```java
    // Test invalid date formats
    @Test
    public void testInvalidDateInvalidMonth() { //Tests an invalid month value
        assertFalse(regexValidator.validDate(theDate:"13-01-2023"));
    }

    @Test
    public void testInvalidDateInvalidDay() { //Tests an invalid day value
        assertFalse(regexValidator.validDate(theDate:"02-30-2023"));
    }

    @Test
    public void testInvalidDateInvalidLeapYear() { //Tests an invalid leap year date
        assertFalse(regexValidator.validDate(theDate:"02-29-2023"));
    }

    @Test
    public void testInvalidDateWrongSeparator() { //Tests an incorrect separator
        assertFalse(regexValidator.validDate(theDate:"12.31.2023"));
    }

    @Test
    public void testInvalidDateNonNumeric() { //Tests a date containing non-numeric characters
        assertFalse(regexValidator.validDate(theDate:"12-AB-2023"));
    }

    @Test
    public void testInvalidDateEmptyString() { //Tests an empty string
        assertFalse(regexValidator.validDate(theDate:""));
    }

    @Test
    public void testInvalidDateMissingComponents() { //Tests a date missing components
        assertFalse(regexValidator.validDate(theDate:"12-2023"));
    }

    @Test
    public void testInvalidDateOutOfRange() { //Tests a date with out-of-range values
        assertFalse(regexValidator.validDate(theDate:"00-00-0000"));
    }
}

```

6. House Address:

```java
import org.junit.jupiter.api.Test;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

public class AddressValidationTest {

    // Test valid addresses
    @Test
    public void testValidAddressStandardFormat() { //Tests a standard address format
        assertTrue(regexValidator.validAddress(theAddress:"123 Main Street"));
    }

    @Test
    public void testValidAddressWithAbbreviation() { //Tests abbreviation of street types
        assertTrue(regexValidator.validAddress(theAddress:"456 Maple Ave"));
    }

    @Test
    public void testValidAddressWithFullWordForStreetType() { //Tests full street type name (Boulevard)
        assertTrue(regexValidator.validAddress(theAddress:"789 Oak Boulevard"));
    }

    @Test
    public void testValidAddressWithDirectionalPrefix() { //Tests address with a directional prefix
        assertTrue(regexValidator.validAddress(theAddress:"321 North Washington Street"));
    }

    @Test
    public void testValidAddressWithMultiWordStreetName() { //Tests street name with multiple words
        assertTrue(regexValidator.validAddress(theAddress:"654 Martin Luther King Avenue"));
    }

    @Test
    public void testValidAddressWithApartmentNumber() { //Tests address containing an apartment number
        assertTrue(regexValidator.validAddress(theAddress:"987 Pine Street Apt 4B"));
    }

    @Test
    public void testValidAddressWithUnitNumber() { //Tests address containing a unit number
        assertTrue(regexValidator.validAddress(theAddress:"246 Cedar Road Unit 7"));
    }

    @Test
    public void testValidAddressWithDirectionalAndAbbreviation() { //Tests address with a directional prefix and an abbrevi
        assertTrue(regexValidator.validAddress(theAddress:"135 South Park Blvd"));
    }

```

```
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:

```java
import org.junit.jupiter.api.Test;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

public class LocationValidationTest {

    @Test
    public void testValidStandardCityStateZip() {
        assertTrue(regexValidator.validLocation(theLocation:"Seattle, WA 98101")); //Test Valid city, state, zip format
    }

    @Test
    public void testValidCityStateZipWithMultiWordCity() {
        assertTrue(regexValidator.validLocation(theLocation:"San Francisco, CA 94105")); //Test Multi-word city
    }

    @Test
    public void testValidCityStateZipWithHyphenatedCity() {
        assertTrue(regexValidator.validLocation(theLocation:"Ann Arbor, MI 48104")); //Test Hyphenated city
    }

    @Test
    public void testValidCityStateZipWithExtendedZip() {
        assertTrue(regexValidator.validLocation(theLocation:"Portland, OR 97201-1234")); //Test Extended zip code
    }

    @Test
    public void testValidCityStateZipWithSpaceInState() {
        assertTrue(regexValidator.validLocation(theLocation:"New York, NY 10001")); //Test Space in state abbreviation
    }

    @Test
    public void testValidCityStateZipLowercaseState() {
        assertTrue(regexValidator.validLocation(theLocation:"Chicago, IL 60601")); //Test Lowercase state
    }

    @Test
    public void testValidCityWithApostrophe() {
        assertTrue(regexValidator.validLocation(theLocation:"O'Fallon, MO 63366")); //Test Apostrophe in city
    }

    @Test
    public void testValidCityStateZipWithAlternateSpacing() {
        assertTrue(regexValidator.validLocation(theLocation:"Denver,CO 80202")); //Test Alternate spacing
    }

```

```java
    @Test
    public void testInvalidCityStateZipMissingState() {
        assertFalse(regexValidator.validLocation(theLocation:"Seattle 98101")); //Test Missing state
    }

    @Test
    public void testInvalidCityStateZipInvalidStateAbbreviation() {
        assertFalse(regexValidator.validLocation(theLocation:"Seattle, XX 98101")); //Test Invalid state abbreviation
    }

    @Test
    public void testInvalidCityStateZipWrongOrder() {
        assertFalse(regexValidator.validLocation(theLocation:"WA Seattle 98101")); //Test Wrong order
    }

    @Test
    public void testInvalidCityStateZipMissingZip() {
        assertFalse(regexValidator.validLocation(theLocation:"Seattle, WA")); //Test Missing zip code
    }

    @Test
    public void testInvalidCityStateZipInvalidZipFormat() {
        assertFalse(regexValidator.validLocation(theLocation:"Seattle, WA 9810")); //Test Invalid zip format
    }

    @Test
    public void testInvalidCityStateZipSpecialCharacters() {
        assertFalse(regexValidator.validLocation(theLocation:"Seattle! WA 98101")); //Test Special characters in city
    }

    @Test
    public void testInvalidCityStateZipNumericCity() {
        assertFalse(regexValidator.validLocation(theLocation:"123, WA 98101")); //Test Numeric city
    }

    @Test
    public void testInvalidCityStateZipEmptyString() {
        assertFalse(regexValidator.validLocation(theLocation:"")); //Test Empty string
    }
}

```

8. Military Time:

```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class MilitaryTimeValidationTest {
    @Test
    public void testValidMilitaryTimeMidnight() { // Test valid military time for midnight (0000)
        assertTrue(regexValidator.validMilitaryTime(theTime:"0000"));
    }

    @Test
    public void testValidMilitaryTimeNoon() { // Test valid military time for noon (1200)
        assertTrue(regexValidator.validMilitaryTime(theTime:"1200"));
    }

    @Test
    public void testValidMilitaryTimeEarlyMorning() { // Test valid military time for early morning (0530)
        assertTrue(regexValidator.validMilitaryTime(theTime:"0530"));
    }

    @Test
    public void testValidMilitaryTimeSingleDigitHour() { // Test valid military time with a single-digit hour (0900)
        assertTrue(regexValidator.validMilitaryTime(theTime:"0900"));
    }

    @Test
    public void testValidMilitaryTimeFullDay() { // Test valid military time for the last minute of the day (2359)
        assertTrue(regexValidator.validMilitaryTime(theTime:"2359"));
    }

    @Test
    public void testValidMilitaryTimeEveningTime() { // Test valid military time for evening time (1845)
        assertTrue(regexValidator.validMilitaryTime(theTime:"1845"));
    }

    @Test
    public void testValidMilitaryTimeLeadingZero() { // Test valid military time with a leading zero (0005)
        assertTrue(regexValidator.validMilitaryTime(theTime:"0005"));
    }

    @Test
    public void testValidMilitaryTimeSingleDigitMinute() { // Test valid military time with a single-digit minute (1205)
        assertTrue(regexValidator.validMilitaryTime(theTime:"1205"));
    }

```

```java
    @Test
    public void testInvalidMilitaryTimeInvalidHour() { // Test invalid military time with an invalid hour (2460)
        assertFalse(regexValidator.validMilitaryTime(theTime:"2460"));
    }

    @Test
    public void testInvalidMilitaryTimeInvalidMinute() { // Test invalid military time with an invalid minute (1260)
        assertFalse(regexValidator.validMilitaryTime(theTime:"1260"));
    }

    @Test
    public void testInvalidMilitaryTimeNonNumeric() { // Test invalid military time with non-numeric characters (12AB)
        assertFalse(regexValidator.validMilitaryTime(theTime:"12AB"));
    }

    @Test
    public void testInvalidMilitaryTimeTooShort() { // Test invalid military time that is too short (123)
        assertFalse(regexValidator.validMilitaryTime(theTime:"123"));
    }

    @Test
    public void testInvalidMilitaryTimeTooLong() { // Test invalid military time that is too long (123456)
        assertFalse(regexValidator.validMilitaryTime(theTime:"123456"));
    }

    @Test
    public void testInvalidMilitaryTimeWithColon() { // Test invalid military time with a colon (12:30)
        assertFalse(regexValidator.validMilitaryTime(theTime:"12:30"));
    }

    @Test
    public void testInvalidMilitaryTimeEmpty() { // Test invalid military time with an empty string
        assertFalse(regexValidator.validMilitaryTime(theTime:""));
    }

    @Test
    public void testInvalidMilitaryTimeNegative() { // Test invalid military time with a negative sign (-1200)
        assertFalse(regexValidator.validMilitaryTime(theTime:"-1200"));
    }
}

```

9. US Currency:

```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class CurrencyValidationTest {

    // Test valid currency formats
    @Test
    public void testValidCurrencySimpleValue() { //Tests a simple currency value with two decimal numbers
        assertTrue(regexValidator.validUSCurrency(theCurrency:"$100.00"));
    }

    @Test
    public void testValidCurrencyWithCommas() { //Tests a large currency value with commas
        assertTrue(regexValidator.validUSCurrency(theCurrency:"$1,234,567.89"));
    }

    @Test
    public void testValidCurrencyWholeNumber() { //Tests a whole number without decimal places
        assertTrue(regexValidator.validUSCurrency(theCurrency:"$500"));
    }

    @Test
    public void testValidCurrencyNegativeValue() { //Tests a negative currency value
        assertTrue(regexValidator.validUSCurrency(theCurrency:"-$100.00"));
    }

    @Test
    public void testValidCurrencyLargeNumber() { //Tests a large currency value with commas and decimal places
        assertTrue(regexValidator.validUSCurrency(theCurrency:"$123,456,789.00"));
    }

    @Test
    public void testValidCurrencyCentsOnly() { //Tests a currency value with only cents
        assertTrue(regexValidator.validUSCurrency(theCurrency:"$0.99"));
    }

    @Test
    public void testValidCurrencyZero() { //Tests a zero currency value
        assertTrue(regexValidator.validUSCurrency(theCurrency:"$0.00"));
    }

    @Test
    public void testValidCurrencyWithoutCents() { //Tests a currency value without cents
        assertTrue(regexValidator.validUSCurrency(theCurrency:"$1,000"));
    }

```

```java
    // Test invalid currency formats
    @Test
    public void testInvalidCurrencyMissingDollarSign() { //Tests a missing dollar sign
        assertFalse(regexValidator.validUSCurrency(theCurrency:"100.00"));
    }

    @Test
    public void testInvalidCurrencyInvalidFormat() { //Tests an incorrectly formatted currency value
        assertFalse(regexValidator.validUSCurrency(theCurrency:"$100,00"));
    }

    @Test
    public void testInvalidCurrencyTooManyCents() { //Tests a currency value with more than two decimal places
        assertFalse(regexValidator.validUSCurrency(theCurrency:"$100.001"));
    }

    @Test
    public void testInvalidCurrencyNonNumericChars() { //Tests a currency value containing non-numeric characters
        assertFalse(regexValidator.validUSCurrency(theCurrency:"$10A.00"));
    }

    @Test
    public void testInvalidCurrencyMultipleDollarSigns() { //Tests a currency value with multiple dollar signs
        assertFalse(regexValidator.validUSCurrency(theCurrency:"$$100.00"));
    }

    @Test
    public void testInvalidCurrencyInvalidCommaPlacement() { //Tests an invalid comma placement in the currency value
        assertFalse(regexValidator.validUSCurrency(theCurrency:"$1,00.00"));
    }

    @Test
    public void testInvalidCurrencyEmptyString() { //Tests an empty string
        assertFalse(regexValidator.validUSCurrency(theCurrency:""));
    }

    @Test
    public void testInvalidCurrencySpaces() { //Tests a currency value with spaces after the dollar sign
        assertFalse(regexValidator.validUSCurrency(theCurrency:"$ 100.00"));
    }
}
```

10. URLs:

```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class URLValidationTest {
    @Test
    public void testValidURLHttps() { // Test valid URL with HTTPS protocol
        assertTrue(regexValidator.validURL(theLinks:"https://www.example.com"));
    }

    @Test
    public void testValidURLHttp() { // Test valid URL with HTTP protocol
        assertTrue(regexValidator.validURL(theLinks:"http://example.com"));
    }

    @Test
    public void testValidURLNoProtocol() { // Test valid URL without any protocol
        assertTrue(regexValidator.validURL(theLinks:"www.example.com"));
    }

    @Test
    public void testValidURLWithSubdomain() { // Test valid URL with subdomain
        assertTrue(regexValidator.validURL(theLinks:"https://subdomain.example.co.uk"));
    }

    @Test
    public void testValidURLWithPort() { // Test valid URL with a port number
        assertTrue(regexValidator.validURL(theLinks:"http://localhost:8080"));
    }

    @Test
    public void testValidURLWithPath() { // Test valid URL with a path
        assertTrue(regexValidator.validURL(theLinks:"https://example.com/path/to/page"));
    }

    @Test
    public void testValidURLWithQueryParams() { // Test valid URL with query parameters
        assertTrue(regexValidator.validURL(theLinks:"https://example.com/search?q=test&page=1"));
    }

    @Test
    public void testValidURLMixedCase() { // Test valid URL with mixed case in the domain
        assertTrue(regexValidator.validURL(theLinks:"HTTPS://Example.COM"));
    }

```

```java
    // Invalid URL Tests
    @Test
    public void testInvalidURLMissingDomain() { // Test invalid URL with missing domain
        assertFalse(regexValidator.validURL(theLinks:"https://"));
    }

    @Test
    public void testInvalidURLInvalidProtocol() { // Test invalid URL with an unsupported protocol
        assertFalse(regexValidator.validURL(theLinks:"ftp://example.com"));
    }

    @Test
    public void testInvalidURLMissingDot() { // Test invalid URL missing a dot in the domain name
        assertFalse(regexValidator.validURL(theLinks:"https://examplecom"));
    }

    @Test
    public void testInvalidURLInvalidChars() { // Test invalid URL with invalid characters in the domain
        assertFalse(regexValidator.validURL(theLinks:"https://example!.com"));
    }

    @Test
    public void testInvalidURLSpaces() { // Test invalid URL with spaces in the domain
        assertFalse(regexValidator.validURL(theLinks:"https://example domain.com"));
    }

    @Test
    public void testInvalidURLEmptyString() { // Test invalid URL with an empty string
        assertFalse(regexValidator.validURL(theLinks:""));
    }

    @Test
    public void testInvalidURLOnlyProtocol() { // Test invalid URL with only the protocol and no domain
        assertFalse(regexValidator.validURL(theLinks:"https://"));
    }

    @Test
    public void testInvalidURLInvalidTopLevelDomain() { // Test invalid URL with an invalid top-level domain
        assertFalse(regexValidator.validURL(theLinks:"https://example.invalidtld"));
    }
}

```

11. Password:

```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class PassWordValidationTest {

    @Test
    public void testValidPasswordMinimumLength() {
        assertTrue(regexValidator.validPassword(thePassword:"Abc!123456")); // Exactly 10 characters, meets all conditions
    }

    @Test
    public void testValidPasswordExceedsMinimumLength() {
        assertTrue(regexValidator.validPassword(thePassword:"XyZ!45678abc")); // More than 10 characters, still valid
    }

    @Test
    public void testValidPasswordWithMaxConsecutiveLowercase() {
        assertTrue(regexValidator.validPassword(thePassword:"Aabc!123DEF")); // 3 consecutive lowercase allowed
    }

    @Test
    public void testValidPasswordWithMixedCases() {
        assertTrue(regexValidator.validPassword(thePassword:"Pass1!WordX")); // Proper mix of upper, lower, digit, punctu
    }

    @Test
    public void testValidPasswordWithSinglePunctuation() {
        assertTrue(regexValidator.validPassword(thePassword:"Secure!2Pass12")); // Uses exactly one punctuation mark
    }

    @Test
    public void testValidPasswordWithMultipleDigits() {
        assertTrue(regexValidator.validPassword(thePassword:"Strong!9876Abc")); // Extra numbers but still valid
    }

    @Test
    public void testValidPasswordWithDifferentPunctuation() {
        assertTrue(regexValidator.validPassword(thePassword:"Hello!123World")); // '@' as punctuation
    }

    @Test
    public void testValidPasswordWithDifferentPunctuation2() {
        assertTrue(regexValidator.validPassword(thePassword:"XyZ!45678abc")); // '#' as punctuation
    }

```

```java
    @Test
    public void testInvalidPasswordTooShort() {
        assertFalse(regexValidator.validPassword(thePassword:"Ab1!cde")); // Less than 10 characters
    }

    @Test
    public void testInvalidPasswordMissingUppercase() {
        assertFalse(regexValidator.validPassword(thePassword:"abcdef!1234")); // No uppercase letter
    }

    @Test
    public void testInvalidPasswordMissingLowercase() {
        assertFalse(regexValidator.validPassword(thePassword:"ABCDEF!1234")); // No lowercase letter
    }

    @Test
    public void testInvalidPasswordMissingDigit() {
        assertFalse(regexValidator.validPassword(thePassword:"Abcdefgh!X")); // No digit
    }

    @Test
    public void testInvalidPasswordMissingPunctuation() {
        assertFalse(regexValidator.validPassword(thePassword:"Abcdef12345")); // No punctuation
    }

    @Test
    public void testInvalidPasswordTooManyConsecutiveLowercase() {
        assertFalse(regexValidator.validPassword(thePassword:"Abcdeeee!123")); // More than 3 consecutive lowercase letter
    }

    @Test
    public void testInvalidPasswordTooManyConsecutiveLowercase2() {
        assertFalse(regexValidator.validPassword(thePassword:"Abcdaaaa!123")); // More than 3 consecutive lowercase letter
    }

    @Test
    public void testInvalidPasswordWithMultiplePunctuation() {
        assertFalse(regexValidator.validPassword(thePassword:"Secure!Pass@12")); // More than one punctuation mark
    }

```

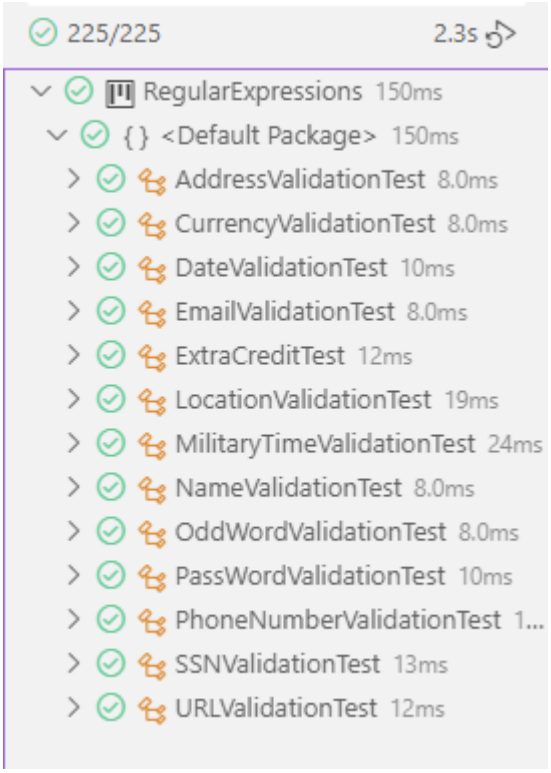
```java
    @Test
    public void testInvalidPasswordWithOnlyLettersNoDigitOrPunctuation() {
        assertFalse(regexValidator.validPassword(thePassword:"Abcdefghijkl")); // No digit, no punctuation
    }

    @Test
    public void testInvalidPasswordWithOnlyDigitsAndNoLetters() {
        assertFalse(regexValidator.validPassword(thePassword:"1234567890!")); // No upper/lowercase letters
    }

    @Test
    public void testInvalidPasswordWithOnlyPunctuationAndLetters() {
        assertFalse(regexValidator.validPassword(thePassword:"Abcdef!!!")); // No digit
    }

    @Test
    public void testInvalidPasswordWithSpaces() {
        assertFalse(regexValidator.validPassword(thePassword:"Abc 123!XYZ")); // Spaces should not be allowed
    }

    @Test
    public void testInvalidPasswordWithTabs() {
        assertFalse(regexValidator.validPassword(thePassword:"Abc\t123!XYZ")); // Tabs should not be allowed
    }

    @Test
    public void testInvalidPasswordWithNewlineCharacters() {
        assertFalse(regexValidator.validPassword(thePassword:"Abc\n123!XYZ")); // Newlines should not be allowed
    }

    @Test
    public void testInvalidPasswordWithTooManyConsecutiveRepeatingCharacters() {
        assertFalse(regexValidator.validPassword(thePassword:"Aaaab!123X")); // More than 3 consecutive 'a' characters
    }

    @Test
    public void testInvalidPasswordWithOnlyUppercaseLetters() {
        assertFalse(regexValidator.validPassword(thePassword:"HELLO123!X")); // No lowercase letter
    }
}
```

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

```java
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

import org.junit.Test;

public class OddWordValidationTest {

    @Test
    public void testValidOddWordShortest() {
        assertTrue(regexValidator.validlengthWord(word:"ion")); // 3 total letters (odd)
    }

    @Test
    public void testValidOddWordFiveLetters() {
        assertTrue(regexValidator.validlengthWord(word:"union")); // 5 total letters (odd)
    }

    @Test
    public void testValidOddWordSevenLetters() {
        assertTrue(regexValidator.validlengthWord(word:"invention")); // 7 total letters (odd)
    }

    @Test
    public void testValidOddWordMixedCase() {
        assertTrue(regexValidator.validlengthWord(word:"Suppression")); // Case insensitive check
    }

    @Test
    public void testValidOddWordThirteenLetters() {
        assertTrue(regexValidator.validlengthWord(word:"participation")); // 13 total letters (odd)
    }

    @Test
    public void testValidOddWordNineteenLetters() {
        assertTrue(regexValidator.validlengthWord(word:"conceptualization")); // 19 total letters (odd)
    }

    @Test
    public void testValidOddWordTwentyThreeLetters() {
        assertTrue(regexValidator.validlengthWord(word:"Overintellectualization")); // 23 total letters (odd)
    }

    @Test
    public void testValidOddWordWithCapitalization() {
        assertTrue(regexValidator.validlengthWord(word:"Commercialization")); // Should be case insensitive
    }
```

```java
    @Test
    public void testInvalidWordWrongEnding() {
        assertFalse(regexValidator.validlengthWord(word:"motivation")); // not correct length (even)
    }

    @Test
    public void testInvalidWordWithNumbers() {
        assertFalse(regexValidator.validlengthWord(word:"m0tion")); // Contains a number
    }

    @Test
    public void testInvalidWordWithSpecialCharacters() {
        assertFalse(regexValidator.validlengthWord(word:"moti@nion")); // Contains special character
    }

    @Test
    public void testInvalidWordWithSpaces() {
        assertFalse(regexValidator.validlengthWord(word:" expulsion ")); // Leading/trailing spaces
    }

    @Test
    public void testInvalidWordWithIncorrectPattern() {
        assertFalse(regexValidator.validlengthWord(word:"bahion")); // Doesn't follow the regex pattern
    }

    @Test
    public void testInvalidWordEvenLengthFourLetters() {
        assertFalse(regexValidator.validlengthWord(word:"xion")); // 4 letters (even)
    }

    @Test
    public void testInvalidWordEvenLengthFourteenLetters() {
        assertFalse(regexValidator.validlengthWord(word:"disintegration")); // 14 total letters (even)
    }

    @Test
    public void testInvalidWordNoIonEnding() {
        assertFalse(regexValidator.validlengthWord(word:"apple")); // Doesn't end in "ion"
    }
}
```

All test passed:



Extra Credit for all 3:

```java
import org.junit.Test;
import static org.junit.Assert.*;

public class ExtraCreditTest {
    // SSN Extra Credit Tests - Testing SSA numbering rules
    @Test
    public void testInvalidSSNArea666() {
        assertFalse(regexValidator.validSSN(theSSN:"666-45-6789"));
    }

    @Test
    public void testInvalidSSNArea000() {
        assertFalse(regexValidator.validSSN(theSSN:"000-45-6789"));
    }

    @Test
    public void testInvalidSSNArea900() {
        assertFalse(regexValidator.validSSN(theSSN:"900-45-6789"));
    }

    @Test
    public void testInvalidSSNArea999() {
        assertFalse(regexValidator.validSSN(theSSN:"999-45-6789"));
    }

    @Test
    public void testValidSSNAreaNumber() {
        assertTrue(regexValidator.validSSN(theSSN:"123-45-6789"));
    }

    @Test
    public void testInvalidSSNGroup00() {
        assertFalse(regexValidator.validSSN(theSSN:"123-00-6789"));
    }

    @Test
    public void testInvalidSSNSerial0000() {
        assertFalse(regexValidator.validSSN(theSSN:"123-45-0000"));
    }

    @Test
    public void testValidSSNBoundaryArea() {
        assertTrue(regexValidator.validSSN(theSSN:"899-45-6789"));
    }
```

```java
    // Phone Number Extra Credit Tests - Testing official area codes
    @Test
    public void testValidAreaCode212() {
        assertTrue(regexValidator.validPhoneNumber(thePhoneNumber:"(212)555-1234")); // NYC
    }

    @Test
    public void testValidAreaCode415() {
        assertTrue(regexValidator.validPhoneNumber(thePhoneNumber:"(415)555-1234")); // San Francisco
    }

    @Test
    public void testValidAreaCode305() {
        assertTrue(regexValidator.validPhoneNumber(thePhoneNumber:"(305)555-1234")); // Miami
    }

    @Test
    public void testInvalidAreaCode000() {
        assertFalse(regexValidator.validPhoneNumber(thePhoneNumber:"(000)555-1234"));
    }

    @Test
    public void testInvalidAreaCode001() {
        assertFalse(regexValidator.validPhoneNumber(thePhoneNumber:"(001)555-1234"));
    }

    @Test
    public void testInvalidAreaCode123() {
        assertFalse(regexValidator.validPhoneNumber(thePhoneNumber:"(123)555-1234")); // Invalid area code
    }

    @Test
    public void testInvalidAreaCode999() {
        assertFalse(regexValidator.validPhoneNumber(thePhoneNumber:"(999)555-1234"));
    }

    @Test
    public void testInvalidAreaCodeAlpha() {
        assertFalse(regexValidator.validPhoneNumber(thePhoneNumber:"(ABC)555-1234"));
    }
```

```java
    // State Abbreviation Extra Credit Tests
    @Test
    public void testValidStateWA() {
        assertTrue(regexValidator.validLocation(theLocation:"Seattle, WA 98101"));
    }

    @Test
    public void testValidStateNY() {
        assertTrue(regexValidator.validLocation(theLocation:"New York, NY 10001"));
    }

    @Test
    public void testValidStateCA() {
        assertTrue(regexValidator.validLocation(theLocation:"Los Angeles, CA 90001"));
    }

    @Test
    public void testInvalidStateXX() {
        assertFalse(regexValidator.validLocation(theLocation:"Invalid, XX 12345"));
    }

    @Test
    public void testInvalidState00() {
        assertFalse(regexValidator.validLocation(theLocation:"Invalid, 00 12345"));
    }

    @Test
    public void testInvalidStateLowercase() {
        assertFalse(regexValidator.validLocation(theLocation:"Invalid, wa 12345"));
    }

    @Test
    public void testInvalidStateMixedCase() {
        assertFalse(regexValidator.validLocation(theLocation:"Invalid, Wa 12345"));
    }

    @Test
    public void testInvalidStateThreeLetters() {
        assertFalse(regexValidator.validLocation(theLocation:"Invalid, WAA 12345"));
    }
}
```