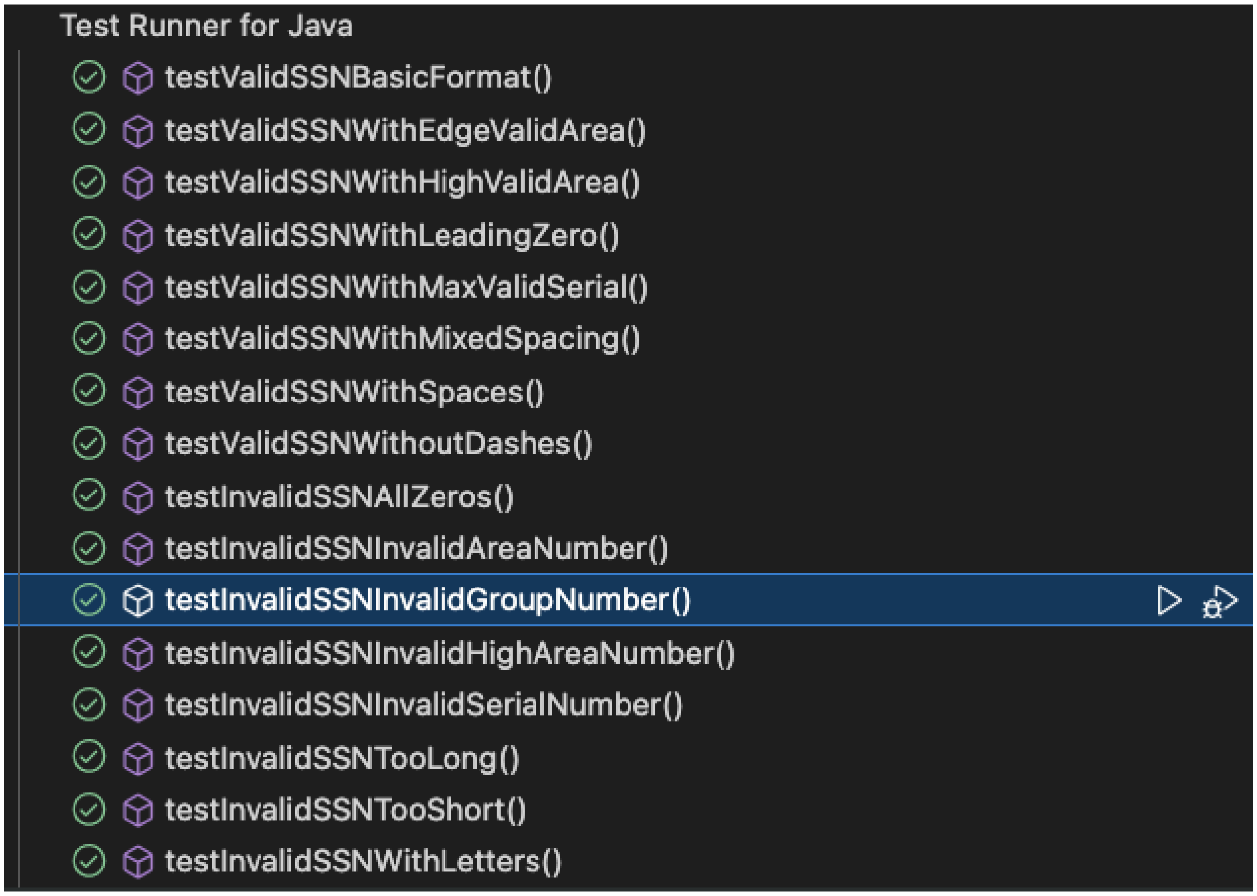
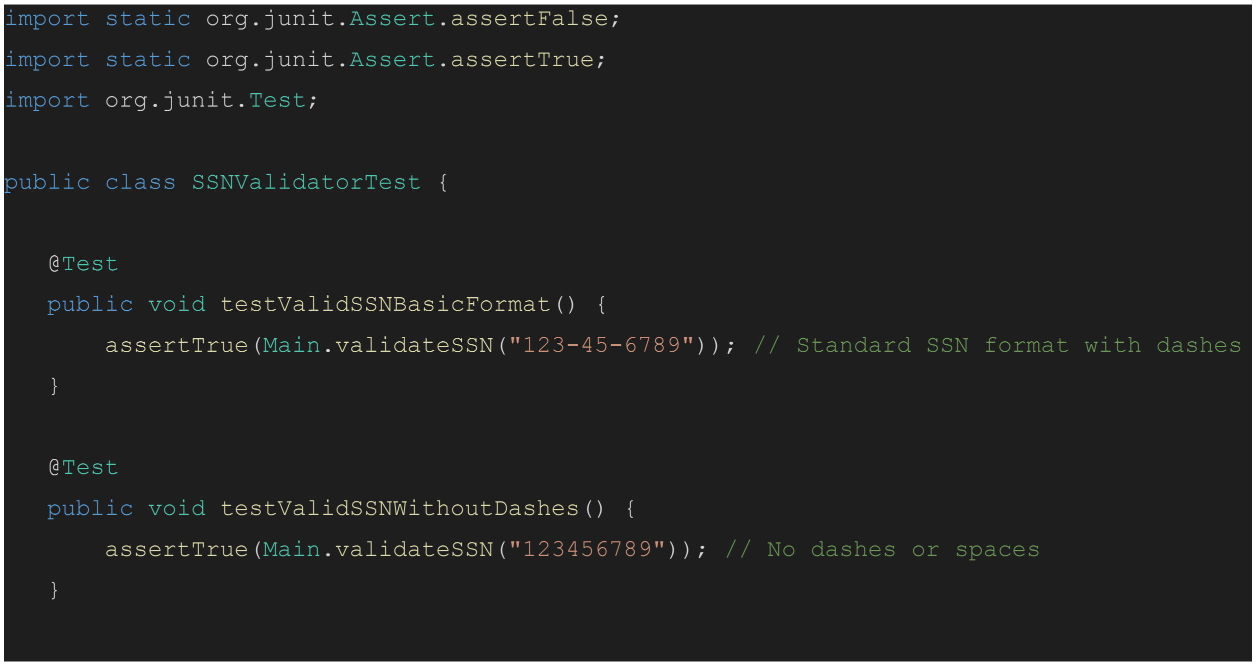
Social Security Number (can be with dashes, whitespace, or no spaces at all)
2 points Extra credit if you make sure the SSN is allowable based on numbering rules by the Social Security Administration



```
Test Runner for Java
✓ testValidSSNBasicFormat()
✓ testValidSSNWithEdgeValidArea()
✓ testValidSSNWithHighValidArea()
✓ testValidSSNWithLeadingZero()
✓ testValidSSNWithMaxValidSerial()
✓ testValidSSNWithMixedSpacing()
✓ testValidSSNWithSpaces()
✓ testValidSSNWithoutDashes()
✓ testInvalidSSNAllZeros()
✓ testInvalidSSNInvalidAreaNumber()
✓ testInvalidSSNInvalidGroupNumber()
✓ testInvalidSSNInvalidHighAreaNumber()
✓ testInvalidSSNInvalidSerialNumber()
✓ testInvalidSSNTooLong()
✓ testInvalidSSNTooShort()
✓ testInvalidSSNWithLetters()
```

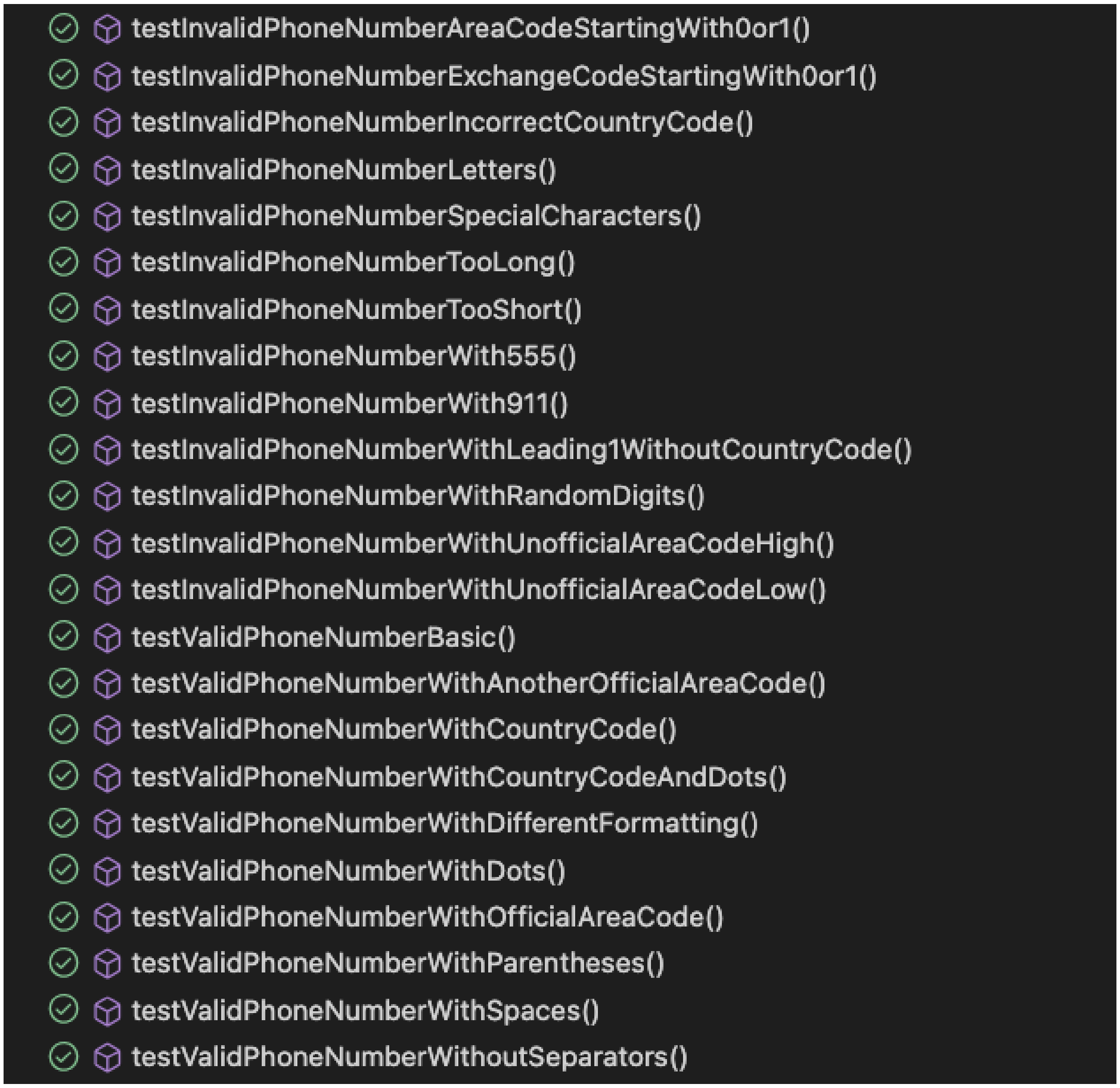


```java
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;
import org.junit.Test;

public class SSNValidatorTest {

   @Test
   public void testValidSSNBasicFormat() {
       assertTrue(Main.validateSSN("123-45-6789")); // Standard SSN format with dashes
   }

   @Test
   public void testValidSSNWithoutDashes() {
       assertTrue(Main.validateSSN("123456789")); // No dashes or spaces
   }
```

```java
    @Test
    public void testValidSSNWithSpaces() {
        assertTrue(Main.validateSSN("123 45 6789")); // Spaces instead of dashes
    }

    @Test
    public void testValidSSNWithMixedSpacing() {
        assertTrue(Main.validateSSN("123-45 6789")); // Mixed dashes and spaces
    }

    @Test
    public void testValidSSNWithLeadingZero() {
        assertTrue(Main.validateSSN("012-34-5678")); // Leading zero in the area number
    }

    // EXTRA CREDIT TEST - SSA Valid Area Number (Highest Valid)
    @Test
    public void testValidSSNWithHighValidArea() {
        assertTrue(Main.validateSSN("899-12-3456")); // Highest valid area number under 900
    }

    //  EXTRA CREDIT TEST - SSA Valid Area Number (Lowest Valid)
    @Test
    public void testValidSSNWithEdgeValidArea() {
        assertTrue(Main.validateSSN("001-23-4567")); // Lowest valid area number
    }

    //  EXTRA CREDIT TEST - SSA Valid Serial Number (Max Valid)
    @Test
    public void testValidSSNWithMaxValidSerial() {
        assertTrue(Main.validateSSN("123-45-9999")); // Valid last four digits at max range
    }

    @Test
    public void testInvalidSSNAllZeros() {
        assertFalse(Main.validateSSN("000-00-0000")); // Completely invalid SSN
    }

    //  EXTRA CREDIT TEST - SSA Invalid Area Number (666)
    @Test
    public void testInvalidSSNInvalidAreaNumber() {
        assertFalse(Main.validateSSN("666-12-3456")); // 666 is not a valid area number
    }

    //  EXTRA CREDIT TEST - SSA Invalid Area Number (900-999)
    @Test
    public void testInvalidSSNInvalidHighAreaNumber() {
        assertFalse(Main.validateSSN("999-12-3456")); // Area numbers 900-999 are not
valid
    }

    //  EXTRA CREDIT TEST - SSA Invalid Group Number (00)
    @Test
    public void testInvalidSSNInvalidGroupNumber() {
        assertFalse(Main.validateSSN("123-00-4567")); // Middle two digits cannot be 00
    }

    //  EXTRA CREDIT TEST - SSA Invalid Serial Number (0000)
    @Test
    public void testInvalidSSNInvalidSerialNumber() {
        assertFalse(Main.validateSSN("123-45-0000")); // Last four digits cannot be
0000
    }

    @Test
    public void testInvalidSSNTooShort() {
        assertFalse(Main.validateSSN("123-45-678")); // Missing a digit
    }

    @Test
    public void testInvalidSSNTooLong() {
        assertFalse(Main.validateSSN("123-45-67890")); // Extra digit
    }

    @Test
    public void testInvalidSSNWithLetters() {
        assertFalse(Main.validateSSN("123-AB-6789")); // Contains non-numeric
characters
    }
}
```

US Phone number - parentheses are optional, as well as the dash between the last two sections

2 points Extra credit if you allow only official area codes

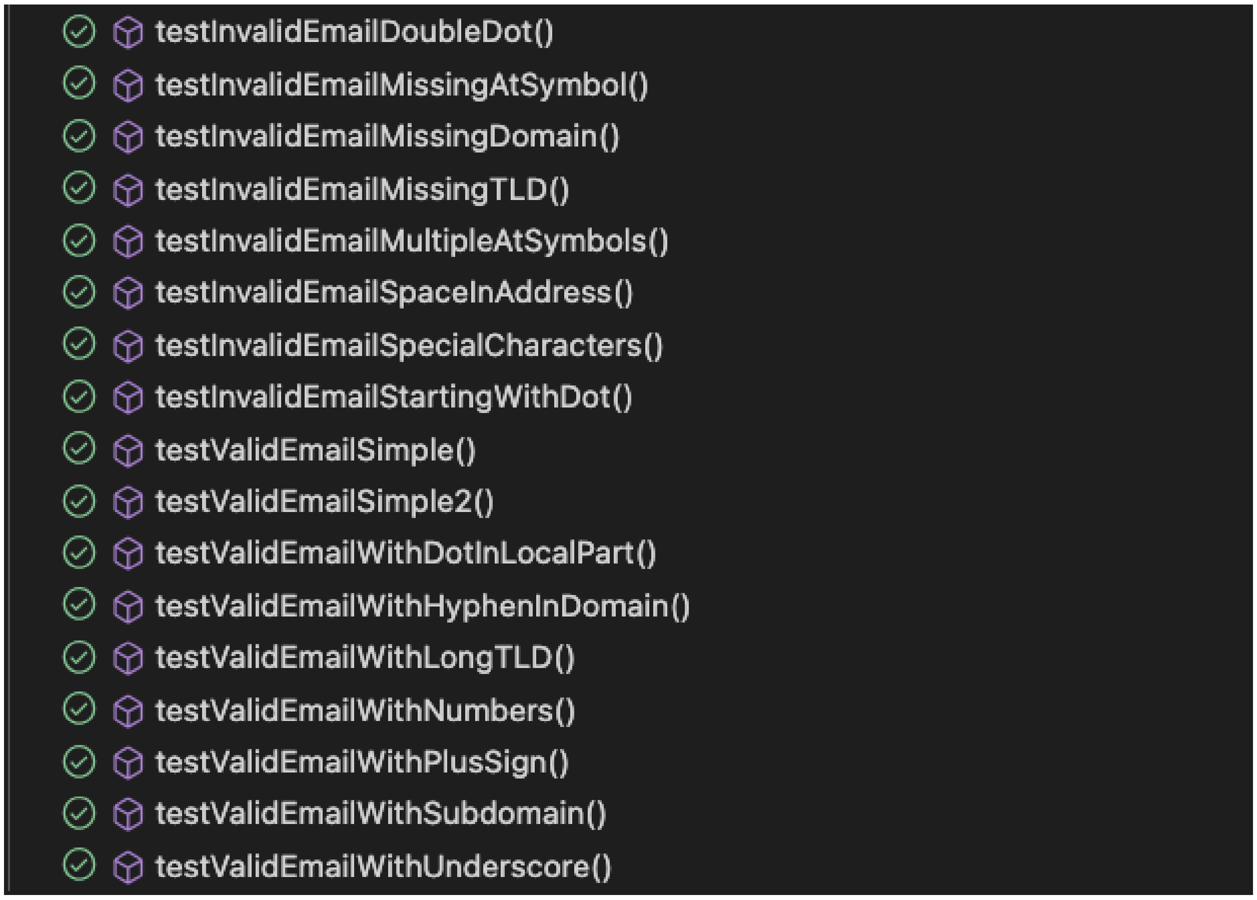


```
✓ testInvalidPhoneNumberAreaCodeStartingWith0or1()
✓ testInvalidPhoneNumberExchangeCodeStartingWith0or1()
✓ testInvalidPhoneNumberIncorrectCountryCode()
✓ testInvalidPhoneNumberLetters()
✓ testInvalidPhoneNumberSpecialCharacters()
✓ testInvalidPhoneNumberTooLong()
✓ testInvalidPhoneNumberTooShort()
✓ testInvalidPhoneNumberWith555()
✓ testInvalidPhoneNumberWith911()
✓ testInvalidPhoneNumberWithLeading1WithoutCountryCode()
✓ testInvalidPhoneNumberWithRandomDigits()
✓ testInvalidPhoneNumberWithUnofficialAreaCodeHigh()
✓ testInvalidPhoneNumberWithUnofficialAreaCodeLow()
✓ testValidPhoneNumberBasic()
✓ testValidPhoneNumberWithAnotherOfficialAreaCode()
✓ testValidPhoneNumberWithCountryCode()
✓ testValidPhoneNumberWithCountryCodeAndDots()
✓ testValidPhoneNumberWithDifferentFormatting()
✓ testValidPhoneNumberWithDots()
✓ testValidPhoneNumberWithOfficialAreaCode()
✓ testValidPhoneNumberWithParentheses()
✓ testValidPhoneNumberWithSpaces()
✓ testValidPhoneNumberWithoutSeparators()
```

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

public class PhoneNumberValidatorTest {


   @Test
   public void testValidPhoneNumberBasic() {
       assertTrue(Main.validatePhoneNumber("678-707-9460")); // Standard format
   }

   @Test
   public void testValidPhoneNumberWithDots() {
       assertTrue(Main.validatePhoneNumber("312.456.7890")); // Dots as separators
   }

   @Test
   public void testValidPhoneNumberWithSpaces() {
       assertTrue(Main.validatePhoneNumber("415 456 7890")); // Spaces as separators
   }

   @Test
   public void testValidPhoneNumberWithParentheses() {
       assertTrue(Main.validatePhoneNumber("(305) 456-7890")); // Parentheses around
area code
   }

   @Test
   public void testValidPhoneNumberWithCountryCode() {
       assertTrue(Main.validatePhoneNumber("+1 646-456-7890")); // Country code with
space
   }

   @Test
   public void testValidPhoneNumberWithCountryCodeAndDots() {
       assertTrue(Main.validatePhoneNumber("+1.818.456.7890")); // Country code with
dots
   }

   @Test
   public void testValidPhoneNumberWithoutSeparators() {
       assertTrue(Main.validatePhoneNumber("7024567890")); // No separators
   }

   @Test
   public void testValidPhoneNumberWithDifferentFormatting() {
       assertTrue(Main.validatePhoneNumber("+1 (617) 456-7890")); // Mixed formatting
with country code
   }
   @Test
   public void testInvalidPhoneNumberTooShort() {
       assertFalse(Main.validatePhoneNumber("123-456-78")); // Not enough digits
   }

   @Test
   public void testInvalidPhoneNumberTooLong() {
       assertFalse(Main.validatePhoneNumber("123-456-78901")); // Too many digits
   }

   @Test
   public void testInvalidPhoneNumberLetters() {
       assertFalse(Main.validatePhoneNumber("212-456-ABCD")); // Letters in number
   }

   @Test
   public void testInvalidPhoneNumberSpecialCharacters() {
       assertFalse(Main.validatePhoneNumber("212@456#7890")); // Special characters not allowed
   }

   @Test
   public void testInvalidPhoneNumberIncorrectCountryCode() {
       assertFalse(Main.validatePhoneNumber("+99 212-456-7890")); // Invalid country code
   }

   @Test
   public void testInvalidPhoneNumberAreaCodeStartingWith0or1() {
       assertFalse(Main.validatePhoneNumber("(012) 456-7890")); // Area code cannot start with 0
       assertFalse(Main.validatePhoneNumber("(112) 456-7890")); // Area code cannot start with 1
   }

   @Test
   public void testInvalidPhoneNumberExchangeCodeStartingWith0or1() {
       assertFalse(Main.validatePhoneNumber("212-012-7890")); // Exchange code cannot start with 0
       assertFalse(Main.validatePhoneNumber("212-112-7890")); // Exchange code cannot start with 1
   }

   @Test
   public void testValidPhoneNumberWithOfficialAreaCode() {
       assertTrue(Main.validatePhoneNumber("212-456-7890")); //  Extra Credit: NYC
area code (Valid)
   }

   @Test
   public void testValidPhoneNumberWithAnotherOfficialAreaCode() {
       assertTrue(Main.validatePhoneNumber("415-555-1234")); //  Extra Credit: San
Francisco (Valid)
   }

   @Test
   public void testInvalidPhoneNumberWithUnofficialAreaCodeLow() {
       assertFalse(Main.validatePhoneNumber("099-456-7890")); //  Extra Credit: Not a
real area code
   }

   @Test
   public void testInvalidPhoneNumberWithUnofficialAreaCodeHigh() {
       assertFalse(Main.validatePhoneNumber("999-456-7890")); //  Extra Credit: Not a
real area code
   }

   @Test
   public void testInvalidPhoneNumberWith911() {
       assertFalse(Main.validatePhoneNumber("911-456-7890")); //  Extra Credit: 911 is
not a valid area code
   }

   @Test
   public void testInvalidPhoneNumberWith555() {
       assertFalse(Main.validatePhoneNumber("555-456-7890")); //  Extra Credit: 555 is
not a valid area code
   }

   @Test
   public void testInvalidPhoneNumberWithLeading1WithoutCountryCode() {
       assertFalse(Main.validatePhoneNumber("1-800-456-7890")); //  Extra Credit:
Should fail if not formatted properly
```

```
    }

    @Test
    public void testInvalidPhoneNumberWithRandomDigits() {
        assertFalse(Main.validatePhoneNumber("888-123-9999")); //  Extra Credit: Random number, possibly invalid area code
    }
}
```

E-mail address

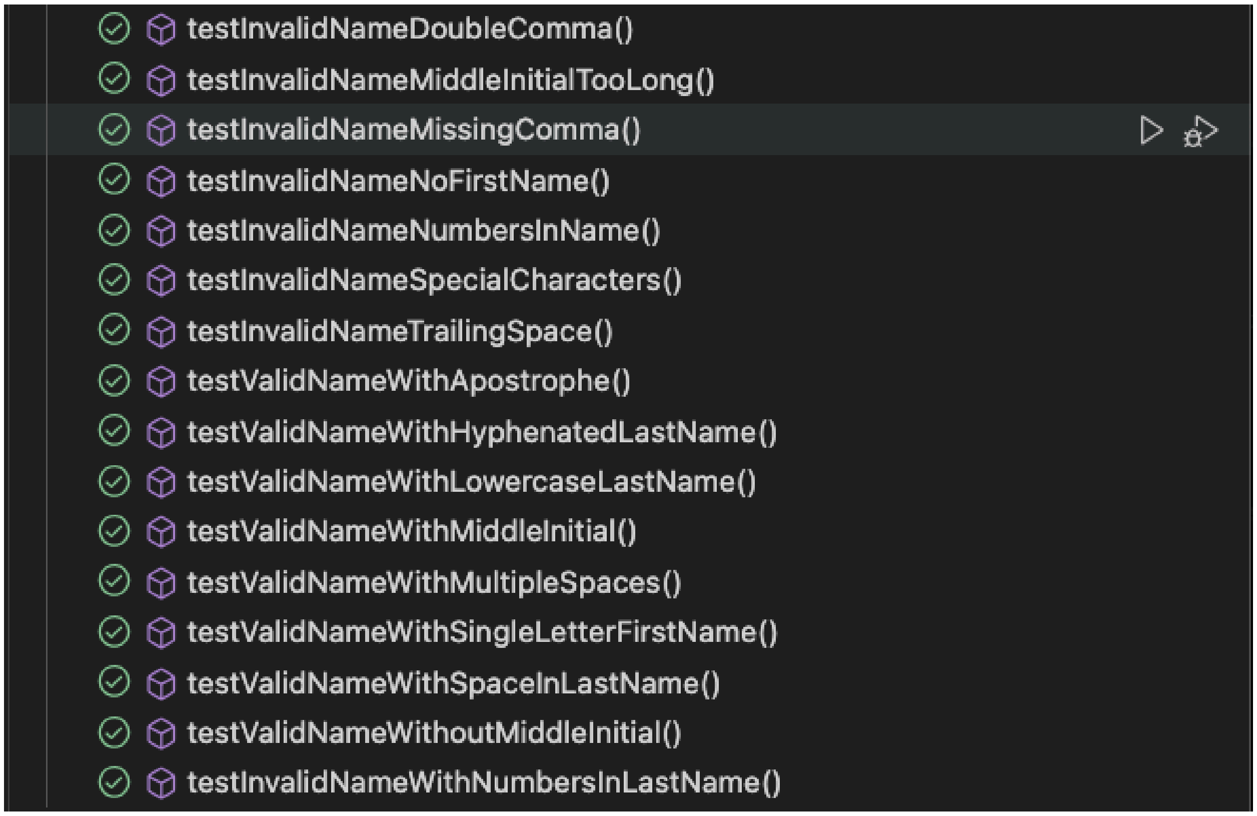


```
✓ testInvalidEmailDoubleDot()
✓ testInvalidEmailMissingAtSymbol()
✓ testInvalidEmailMissingDomain()
✓ testInvalidEmailMissingTLD()
✓ testInvalidEmailMultipleAtSymbols()
✓ testInvalidEmailSpaceInAddress()
✓ testInvalidEmailSpecialCharacters()
✓ testInvalidEmailStartingWithDot()
✓ testValidEmailSimple()
✓ testValidEmailSimple2()
✓ testValidEmailWithDotInLocalPart()
✓ testValidEmailWithHyphenInDomain()
✓ testValidEmailWithLongTLD()
✓ testValidEmailWithNumbers()
✓ testValidEmailWithPlusSign()
✓ testValidEmailWithSubdomain()
✓ testValidEmailWithUnderscore()
```

```
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;
import org.junit.*;
public class EmailValidatorTest {
    @Test
    public void testValidEmailSimple() {
        assertTrue(Main.validateEmail("test@example.com")); // Basic format
    }
    @Test
    public void testValidEmailSimple2() {
        assertTrue(Main.validateEmail("lwazimabota1@gmail.com")); // Basic format
    }

    @Test
    public void testValidEmailWithNumbers() {
        assertTrue(Main.validateEmail("user123@example.com")); // Numbers in local part
    }

    @Test
    public void testValidEmailWithDotInLocalPart() {
        assertTrue(Main.validateEmail("first.last@example.com")); // Dot in local part
    }

    @Test
    public void testValidEmailWithHyphenInDomain() {
        assertTrue(Main.validateEmail("user@my-domain.com")); // Hyphen in domain
    }

    @Test
    public void testValidEmailWithSubdomain() {
        assertTrue(Main.validateEmail("user@mail.example.com")); // Subdomains
    }

    @Test
    public void testValidEmailWithPlusSign() {
        assertTrue(Main.validateEmail("email+alias@gmail.com")); // Plus aliasing
    }

    @Test
    public void testValidEmailWithUnderscore() {
        assertTrue(Main.validateEmail("user_name@example.com")); // Underscore in local
part
    }

    @Test
    public void testValidEmailWithLongTLD() {
        assertTrue(Main.validateEmail("user@example.museum")); // Long TLD
    }

    @Test
    public void testInvalidEmailMissingAtSymbol() {
```

```java
        assertFalse(Main.validateEmail("testexample.com")); // No @ symbol
    }

    @Test
    public void testInvalidEmailMultipleAtSymbols() {
        assertFalse(Main.validateEmail("user@@example.com")); // Multiple @ symbols
    }

    @Test
    public void testInvalidEmailMissingDomain() {
        assertFalse(Main.validateEmail("user@.com")); // No domain name
    }

    @Test
    public void testInvalidEmailMissingTLD() {
        assertFalse(Main.validateEmail("user@example")); // No .TLD
    }

    @Test
    public void testInvalidEmailDoubleDot() {
        assertFalse(Main.validateEmail("user..name@example.com")); // Double dots in
local part
    }

    @Test
    public void testInvalidEmailStartingWithDot() {
        assertFalse(Main.validateEmail(".user@example.com")); // Starts with dot
    }

    @Test
    public void testInvalidEmailSpecialCharacters() {
        assertFalse(Main.validateEmail("user#@example.com")); // Invalid special
character
    }

    @Test
    public void testInvalidEmailSpaceInAddress() {
        assertFalse(Main.validateEmail("user name@example.com")); // Space in email
    }
}
```

Name on a class roster, assuming zero or more middle initials - Last name, First name, MI (e.g. Smith, John, L

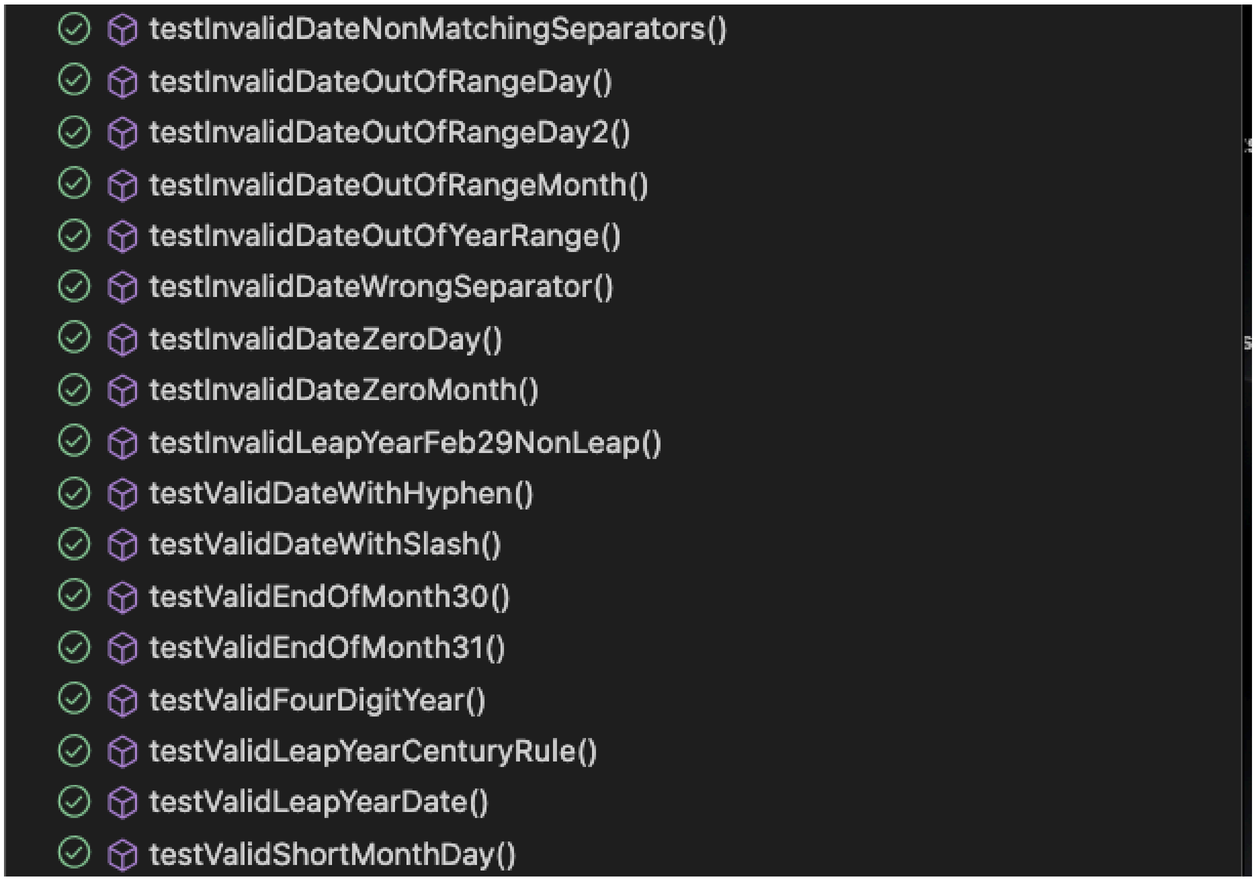


```
✓ testInvalidNameDoubleComma()
✓ testInvalidNameMiddleInitialTooLong()
✓ testInvalidNameMissingComma()
✓ testInvalidNameNoFirstName()
✓ testInvalidNameNumbersInName()
✓ testInvalidNameSpecialCharacters()
✓ testInvalidNameTrailingSpace()
✓ testValidNameWithApostrophe()
✓ testValidNameWithHyphenatedLastName()
✓ testValidNameWithLowercaseLastName()
✓ testValidNameWithMiddleInitial()
✓ testValidNameWithMultipleSpaces()
✓ testValidNameWithSingleLetterFirstName()
✓ testValidNameWithSpaceInLastName()
✓ testValidNameWithoutMiddleInitial()
✓ testInvalidNameWithNumbersInLastName()
```

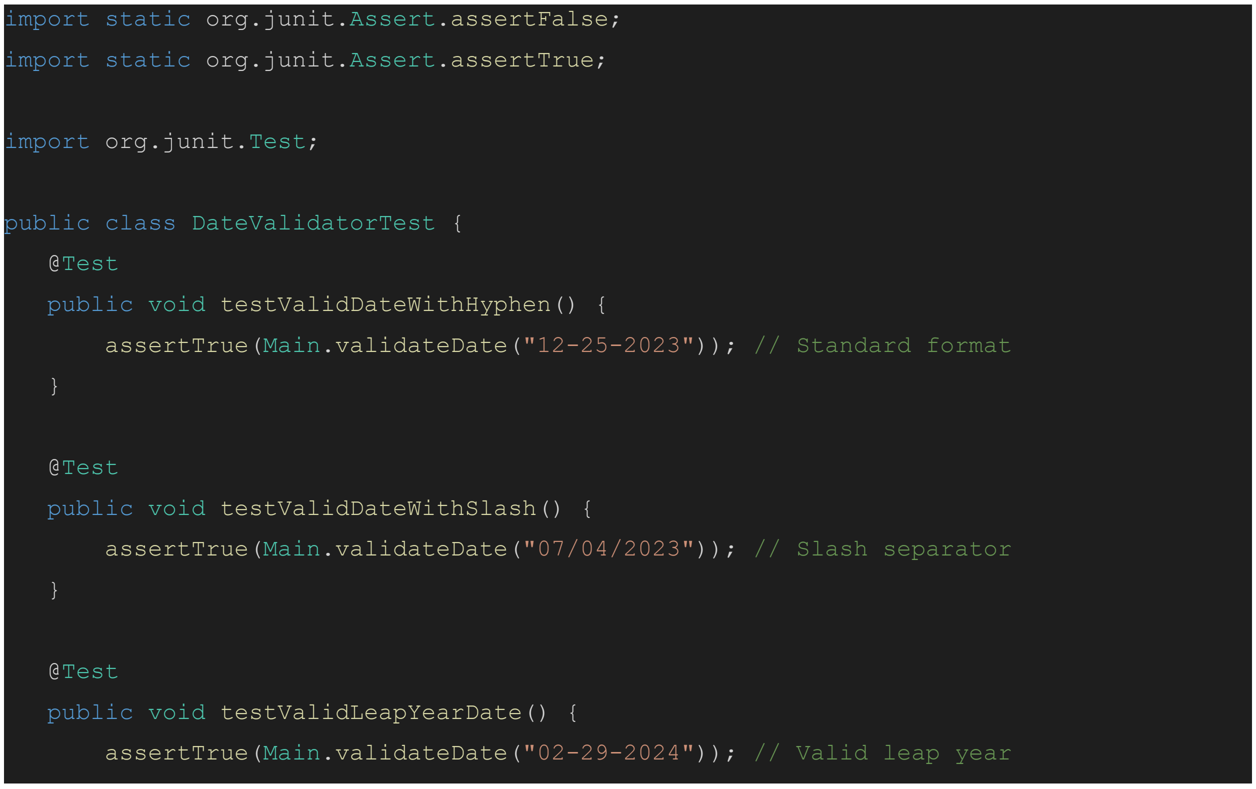
```java
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

import org.junit.Test;

public class NameValidatorTest {
   @Test
   public void testValidNameWithMiddleInitial() {
       assertTrue(Main.validateName("Mabota, Lwazi, M")); // Standard format with MI
   }

   @Test
   public void testValidNameWithoutMiddleInitial() {
       assertTrue(Main.validateName("Doe, Jane")); // No middle initial
   }

   @Test
   public void testValidNameWithHyphenatedLastName() {
       assertTrue(Main.validateName("Smith-Jones, Mary, A")); // Hyphen in last name
   }
```

```
    @Test
    public void testValidNameWithApostrophe() {
        assertTrue(Main.validateName("O'Connor, Patrick, T")); // Apostrophe in last
name
    }

    @Test
    public void testValidNameWithSpaceInLastName() {
        assertTrue(Main.validateName("Van Buren, Martin, V")); // Last name with space
    }

    @Test
    public void testValidNameWithSingleLetterFirstName() {
        assertTrue(Main.validateName("Lee, B, Z")); // Single letter first name
    }

    @Test
    public void testValidNameWithMultipleSpaces() {
        assertTrue(Main.validateName("Jackson,  Michael,   J")); // Extra spaces
    }

    @Test
    public void testValidNameWithLowercaseLastName() {
        assertTrue(Main.validateName("McDonald, Ronald, D")); // Mixed
uppercase/lowercase
    }
    @Test
    public void testInvalidNameMissingComma() {
        assertFalse(Main.validateName("Smith John L")); // No comma
    }

    @Test
    public void testInvalidNameWithNumbersInLastName() {
    assertFalse(Main.validateName("Sm1th, John, L")); // Last name contains a number
    }

    @Test
    public void testInvalidNameNoFirstName() {
        assertFalse(Main.validateName("Doe,")); // No first name
    }
    @Test
    public void testInvalidNameDoubleComma() {
        assertFalse(Main.validateName("Smith,, John, L")); // Extra comma
    }

    @Test
    public void testInvalidNameNumbersInName() {
        assertFalse(Main.validateName("Smith123, John, L")); // Numbers in last name
    }

    @Test
    public void testInvalidNameMiddleInitialTooLong() {
        assertFalse(Main.validateName("Smith, John, ABC")); // More than one middle
initial
    }

    @Test
    public void testInvalidNameSpecialCharacters() {
        assertFalse(Main.validateName("Smith@, John, L")); // Special characters
    }

    @Test
    public void testInvalidNameTrailingSpace() {
        assertFalse(Main.validateName("Smith, John, L ")); // Trailing space
    }
}
```

Date in MM-DD-YYYY format - separators can be -'s, /'s -- you must make sure months, days, year are valid (this includes leap years)



```
✓ testInvalidDateNonMatchingSeparators()
✓ testInvalidDateOutOfRangeDay()
✓ testInvalidDateOutOfRangeDay2()
✓ testInvalidDateOutOfRangeMonth()
✓ testInvalidDateOutOfYearRange()
✓ testInvalidDateWrongSeparator()
✓ testInvalidDateZeroDay()
✓ testInvalidDateZeroMonth()
✓ testInvalidLeapYearFeb29NonLeap()
✓ testValidDateWithHyphen()
✓ testValidDateWithSlash()
✓ testValidEndOfMonth30()
✓ testValidEndOfMonth31()
✓ testValidFourDigitYear()
✓ testValidLeapYearCenturyRule()
✓ testValidLeapYearDate()
✓ testValidShortMonthDay()
```

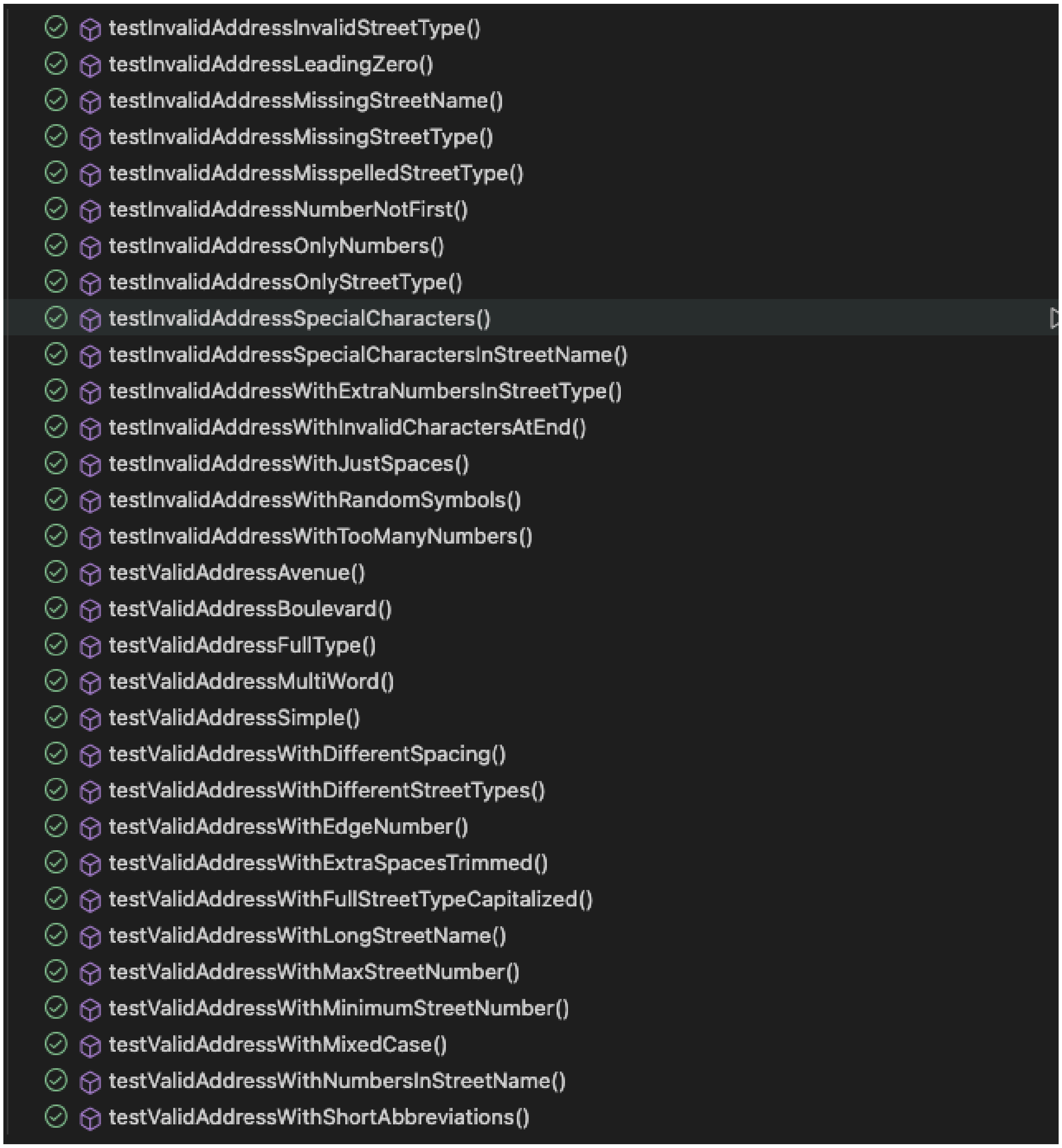


```java
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

import org.junit.Test;

public class DateValidatorTest {
   @Test
   public void testValidDateWithHyphen() {
       assertTrue(Main.validateDate("12-25-2023")); // Standard format
   }

   @Test
   public void testValidDateWithSlash() {
       assertTrue(Main.validateDate("07/04/2023")); // Slash separator
   }

   @Test
   public void testValidLeapYearDate() {
       assertTrue(Main.validateDate("02-29-2024")); // Valid leap year
```

```
    }

    @Test
    public void testValidEndOfMonth30() {
        assertTrue(Main.validateDate("04-30-2023")); // April has 30 days
    }

    @Test
    public void testValidEndOfMonth31() {
        assertTrue(Main.validateDate("12-31-2023")); // December has 31 days
    }

    @Test
    public void testValidShortMonthDay() {
        assertTrue(Main.validateDate("01-01-2023")); // Minimum valid date
    }

    @Test
    public void testValidFourDigitYear() {
        assertTrue(Main.validateDate("06/15/1999")); // Year is in valid range
    }

    @Test
    public void testValidLeapYearCenturyRule() {
        assertTrue(Main.validateDate("02-29-2000")); // 2000 is a leap year (divisible
by 400)
    }

    @Test
    public void testInvalidDateWrongSeparator() {
        assertFalse(Main.validateDate("06.15.2023")); // Invalid separator
    }

    @Test
    public void testInvalidDateNonMatchingSeparators() {
        assertFalse(Main.validateDate("06-15/2023")); // Mixed separators
    }

    @Test
    public void testInvalidDateZeroMonth() {
        assertFalse(Main.validateDate("00-15-2023")); // Month cannot be 00
    }
```

```
   @Test
   public void testInvalidDateZeroDay() {
       assertFalse(Main.validateDate("06-00-2023")); // Day cannot be 00
   }

   @Test
   public void testInvalidDateOutOfRangeMonth() {
       assertFalse(Main.validateDate("13-01-2023")); // Month cannot be 13
   }

   @Test
   public void testInvalidDateOutOfRangeDay() {
       assertFalse(Main.validateDate("04-31-2023")); // April has 30 days
   }

   @Test
   public void testInvalidDateOutOfRangeDay2() {
       assertFalse(Main.validateDate("12-32-2023")); // April has 30 days
   }


   @Test
   public void testInvalidLeapYearFeb29NonLeap() {
       assertFalse(Main.validateDate("02-29-2023")); // 2023 is not a leap year
   }

   @Test
   public void testInvalidDateOutOfYearRange() {
       assertFalse(Main.validateDate("01-15-0999")); // Year must be between 1000 and
9999
   }
}
```

House address - Street number, street name, abbreviation for road, street, boulevard or avenue (full version of those items should also be accepted)



```
testInvalidAddressInvalidStreetType()
testInvalidAddressLeadingZero()
testInvalidAddressMissingStreetName()
testInvalidAddressMissingStreetType()
testInvalidAddressMisspelledStreetType()
testInvalidAddressNumberNotFirst()
testInvalidAddressOnlyNumbers()
testInvalidAddressOnlyStreetType()
testInvalidAddressSpecialCharacters()
testInvalidAddressSpecialCharactersInStreetName()
testInvalidAddressWithExtraNumbersInStreetType()
testInvalidAddressWithInvalidCharactersAtEnd()
testInvalidAddressWithJustSpaces()
testInvalidAddressWithRandomSymbols()
testInvalidAddressWithTooManyNumbers()
testValidAddressAvenue()
testValidAddressBoulevard()
testValidAddressFullType()
testValidAddressMultiWord()
testValidAddressSimple()
testValidAddressWithDifferentSpacing()
testValidAddressWithDifferentStreetTypes()
testValidAddressWithEdgeNumber()
testValidAddressWithExtraSpacesTrimmed()
testValidAddressWithFullStreetTypeCapitalized()
testValidAddressWithLongStreetName()
testValidAddressWithMaxStreetNumber()
testValidAddressWithMinimumStreetNumber()
testValidAddressWithMixedCase()
testValidAddressWithNumbersInStreetName()
testValidAddressWithShortAbbreviations()
```

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

import org.junit.*;
public class AddressValidatorTest {
   @Test
   public void testValidAddressSimple() {
       assertTrue(Main.validateAddress("123 Main St"));
```

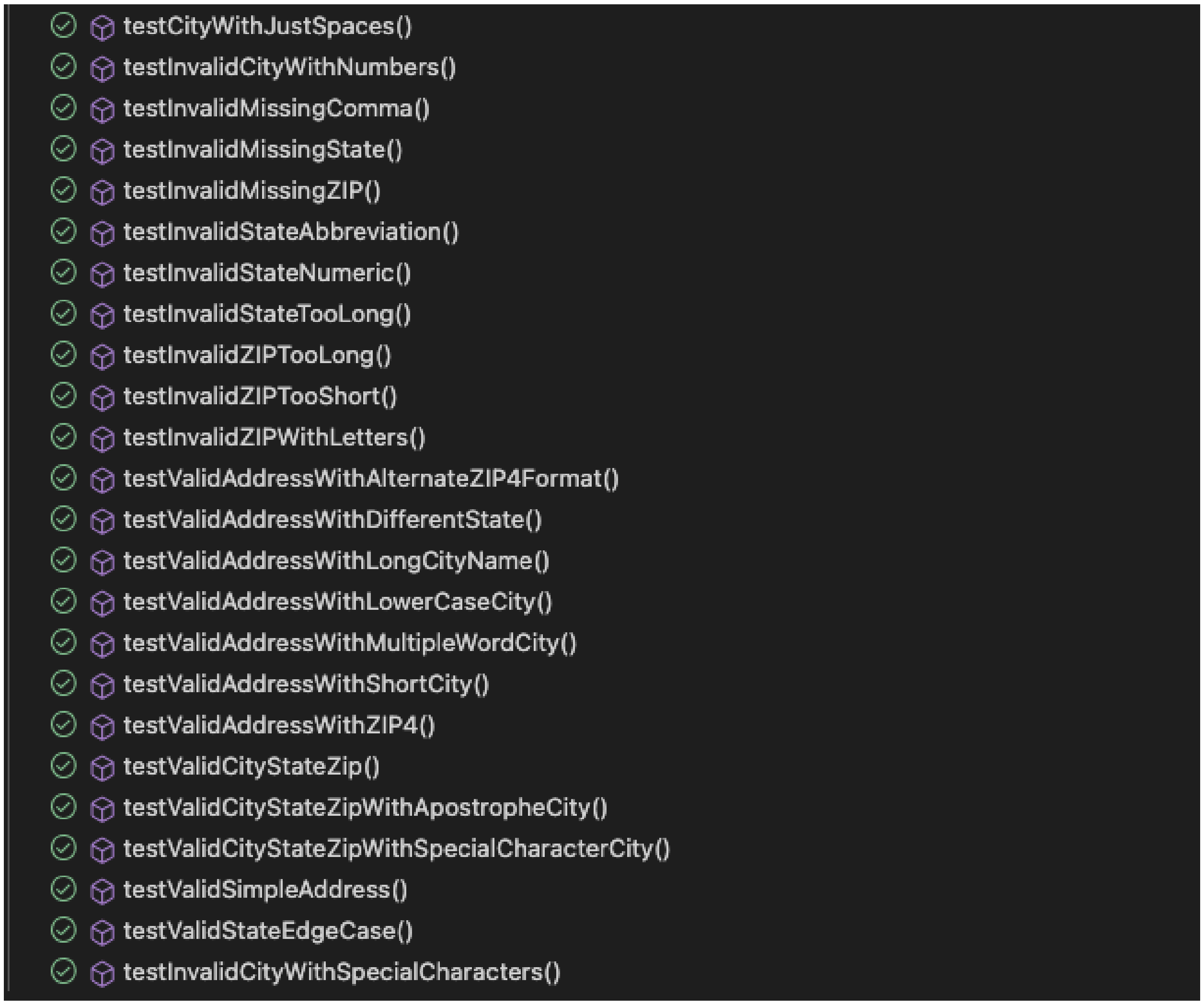
```
    }

    @Test
    public void testValidAddressFullType() {
        assertTrue(Main.validateAddress("456 Oak Road"));
    }

    @Test
    public void testValidAddressBoulevard() {
        assertTrue(Main.validateAddress("789 Sunset Boulevard"));
    }

    @Test
    public void testValidAddressAvenue() {
        assertTrue(Main.validateAddress("101 Pacific Avenue"));
    }

    @Test
    public void testValidAddressMultiWord() {
        assertTrue(Main.validateAddress("222 West Elm Street"));
    }

    @Test
    public void testValidAddressWithMaxStreetNumber() {
        assertTrue(Main.validateAddress("99999 Big Tree Rd"));  // Maximum 5-digit
street number
    }

    @Test
    public void testValidAddressWithMinimumStreetNumber() {
        assertTrue(Main.validateAddress("1 Elm St"));  // Minimum valid street number
    }

    @Test
    public void testValidAddressWithLongStreetName() {
        assertTrue(Main.validateAddress("88 This Is A Very Long Street Name Road"));
    }

    @Test
    public void testInvalidAddressMissingStreetName() {
        assertFalse(Main.validateAddress("123"));
    }
```

```java
@Test
public void testInvalidAddressInvalidStreetType() {
    assertFalse(Main.validateAddress("456 Oak xyz"));
}

@Test
public void testInvalidAddressMissingStreetType() {
    assertFalse(Main.validateAddress("789 Main"));
}

@Test
public void testInvalidAddressLeadingZero() {
    assertFalse(Main.validateAddress("01 Elm St"));
}

@Test
public void testInvalidAddressNumberNotFirst() {
    assertFalse(Main.validateAddress("Main St 123"));
}

@Test
public void testInvalidAddressSpecialCharacters() {
    assertFalse(Main.validateAddress("123*Main*St"));
}


@Test
public void testInvalidAddressMisspelledStreetType() {
    assertFalse(Main.validateAddress("999 Elm Rdd"));
}

@Test
public void testValidAddressWithShortAbbreviations() {
    assertTrue(Main.validateAddress("12 Pine St"));  // "St" abbreviation
}

@Test
public void testValidAddressWithDifferentStreetTypes() {
    assertTrue(Main.validateAddress("345 Maple Blvd"));  // "Blvd" abbreviation
}
```

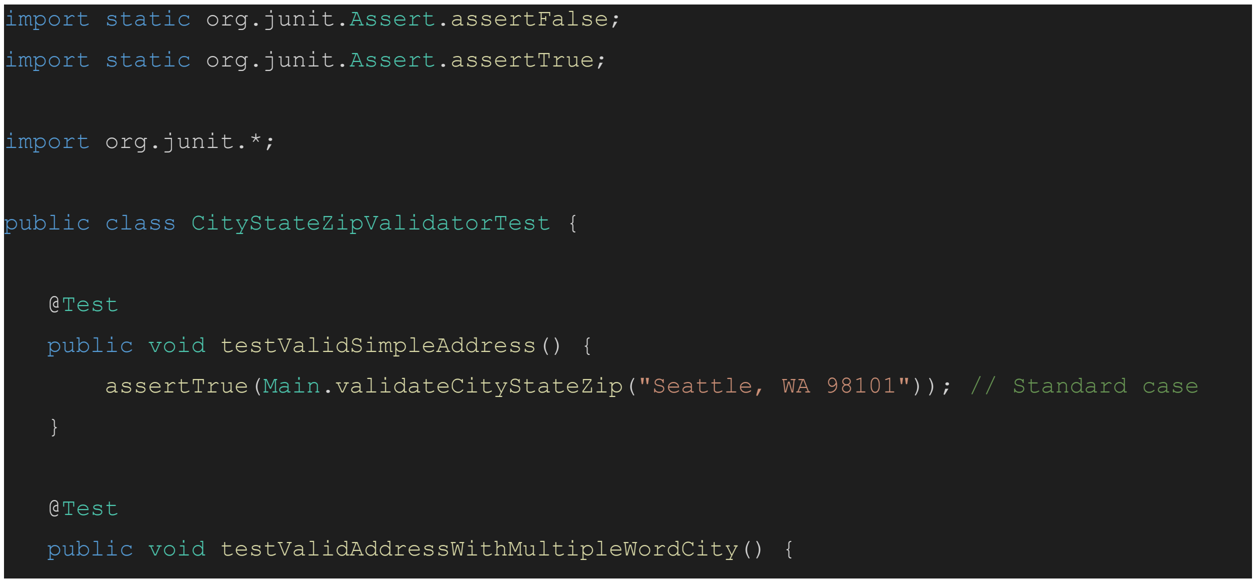
```
   @Test
   public void testValidAddressWithExtraSpacesTrimmed() {
       assertTrue(Main.validateAddress(" 678 Birch Rd  "));  // Should trim spaces
   }

   @Test
   public void testValidAddressWithNumbersInStreetName() {
       assertTrue(Main.validateAddress("400 2nd Avenue"));  // "2nd" should be valid
in name
   }

   @Test
   public void testValidAddressWithMixedCase() {
       assertTrue(Main.validateAddress("890 Elm St"));  // Case-insensitive match
   }

   @Test
   public void testValidAddressWithFullStreetTypeCapitalized() {
       assertTrue(Main.validateAddress("753 Birch STREET"));  // Should accept
uppercase
   }

   @Test
   public void testValidAddressWithDifferentSpacing() {
       assertTrue(Main.validateAddress("951   Oak    Rd"));  // Extra spaces between
words
   }

   @Test
   public void testValidAddressWithEdgeNumber() {
       assertTrue(Main.validateAddress("99999 Grand Ave"));  // Largest possible
street number
   }

   @Test
   public void testInvalidAddressOnlyStreetType() {
       assertFalse(Main.validateAddress("Street"));  // No number or name
   }

   @Test
   public void testInvalidAddressOnlyNumbers() {
       assertFalse(Main.validateAddress("12345"));  // No street name or type
```

```java
    }

    @Test
    public void testInvalidAddressSpecialCharactersInStreetName() {
        assertFalse(Main.validateAddress("101 P!ne St"));  // Invalid character in street name
    }

    @Test
    public void testInvalidAddressWithInvalidCharactersAtEnd() {
        assertFalse(Main.validateAddress("999 Elm Street#"));  // Special character at end
    }

    @Test
    public void testInvalidAddressWithExtraNumbersInStreetType() {
        assertFalse(Main.validateAddress("678 Oak St123"));  // Street type should be at end
    }

    @Test
    public void testInvalidAddressWithRandomSymbols() {
        assertFalse(Main.validateAddress("777 *&^% Blvd"));  // Nonsense characters
    }

    @Test
    public void testInvalidAddressWithJustSpaces() {
        assertFalse(Main.validateAddress("      "));  // Only spaces should fail
    }

    @Test
    public void testInvalidAddressWithTooManyNumbers() {
        assertFalse(Main.validateAddress("123456 Elm St"));  // More than 5 digits
    }
}
```

City followed by state followed by zip as it should appear on a letter
2 points extra credit if you make sure the 2 character state abbreviation (e.g. WA) is valid



```
✓ testCityWithJustSpaces()
✓ testInvalidCityWithNumbers()
✓ testInvalidMissingComma()
✓ testInvalidMissingState()
✓ testInvalidMissingZIP()
✓ testInvalidStateAbbreviation()
✓ testInvalidStateNumeric()
✓ testInvalidStateTooLong()
✓ testInvalidZIPTooLong()
✓ testInvalidZIPTooShort()
✓ testInvalidZIPWithLetters()
✓ testValidAddressWithAlternateZIP4Format()
✓ testValidAddressWithDifferentState()
✓ testValidAddressWithLongCityName()
✓ testValidAddressWithLowerCaseCity()
✓ testValidAddressWithMultipleWordCity()
✓ testValidAddressWithShortCity()
✓ testValidAddressWithZIP4()
✓ testValidCityStateZip()
✓ testValidCityStateZipWithApostropheCity()
✓ testValidCityStateZipWithSpecialCharacterCity()
✓ testValidSimpleAddress()
✓ testValidStateEdgeCase()
✓ testInvalidCityWithSpecialCharacters()
```

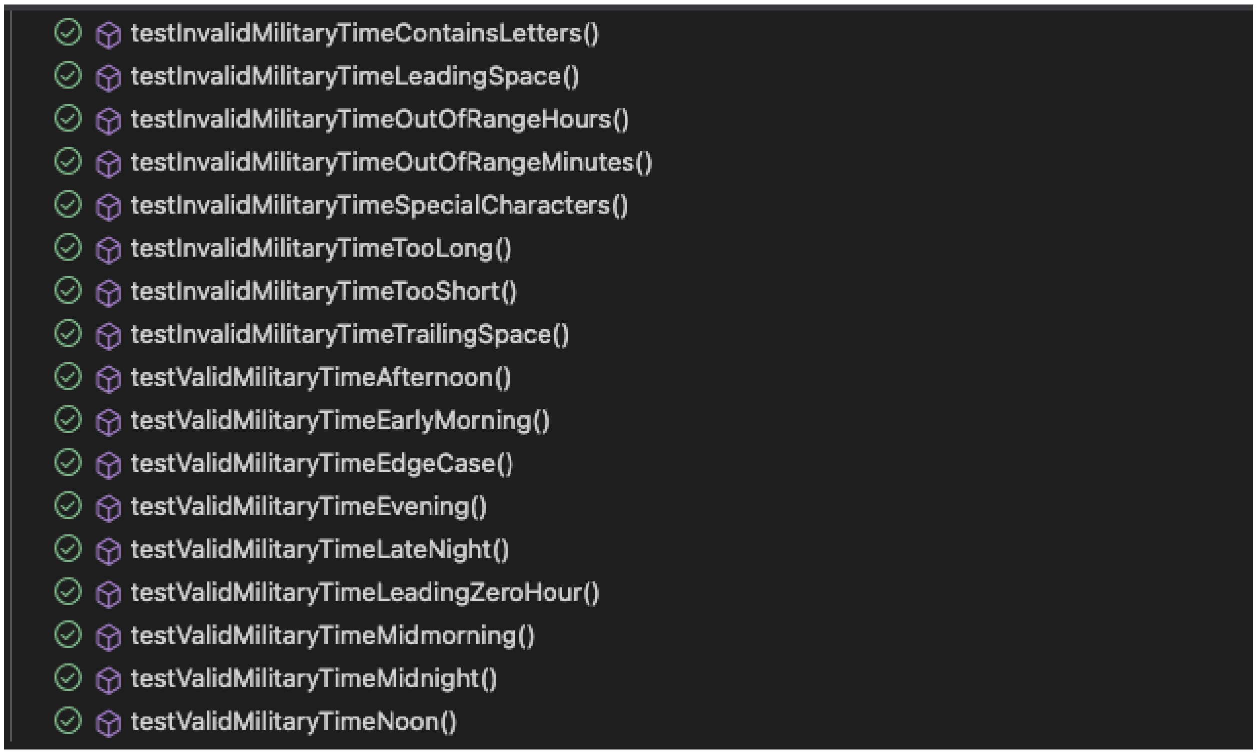


```java
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

import org.junit.*;

public class CityStateZipValidatorTest {

   @Test
   public void testValidSimpleAddress() {
       assertTrue(Main.validateCityStateZip("Seattle, WA 98101")); // Standard case
   }

   @Test
   public void testValidAddressWithMultipleWordCity() {
       assertTrue(Main.validateCityStateZip("San Francisco, CA 94103")); // Multi-word
city name
   }

   @Test
   public void testValidAddressWithZIP4() {
       assertTrue(Main.validateCityStateZip("New York, NY 10001-2345")); // ZIP+4
format
   }

   @Test
   public void testValidAddressWithLowerCaseCity() {
       assertTrue(Main.validateCityStateZip("miami, FL 33101")); // Lowercase city
name (case insensitive)
   }

   @Test
   public void testValidAddressWithDifferentState() {
       assertTrue(Main.validateCityStateZip("Denver, CO 80202")); // Different state
example
   }

   @Test
   public void testValidAddressWithShortCity() {
       assertTrue(Main.validateCityStateZip("Irvine, CA 92602")); // Short city name
   }

   @Test
   public void testValidAddressWithLongCityName() {
       assertTrue(Main.validateCityStateZip("West Palm Beach, FL 33401")); // Long
city name
   }

   @Test
   public void testValidCityStateZip() {
       assertTrue(Main.validateCityStateZip("Chicago, IL 60616")); // Another standard
format
   }

   @Test
   public void testInvalidMissingState() {
       assertFalse(Main.validateCityStateZip("Seattle 98101"));  // No state
   }

   @Test
   public void testInvalidMissingComma() {
       assertFalse(Main.validateCityStateZip("Seattle WA 98101"));  // Missing comma
   }

   @Test
   public void testInvalidMissingZIP() {
       assertFalse(Main.validateCityStateZip("Chicago, IL"));  // No ZIP code
   }

   @Test
   public void testInvalidZIPWithLetters() {
       assertFalse(Main.validateCityStateZip("Houston, TX 77A01"));  // ZIP contains a
letter
   }

   @Test
   public void testInvalidZIPTooShort() {
       assertFalse(Main.validateCityStateZip("Phoenix, AZ 8501"));  // 4-digit ZIP
(should be 5)
   }

   @Test
   public void testInvalidZIPTooLong() {
       assertFalse(Main.validateCityStateZip("Boston, MA 021345"));  // 6-digit ZIP
(invalid)
   }

   @Test
   public void testInvalidCityWithNumbers() {
       assertFalse(Main.validateCityStateZip("123 City, TX 75001"));  // City name has
numbers
   }

   @Test
   public void testCityWithJustSpaces() {
       assertFalse(Main.validateCityStateZip("      "));  // Only spaces should fail
   }
```

```java
   //  **EXTRA CREDIT TEST CASES (State Validation)**

   @Test
   public void testInvalidStateAbbreviation() {
       assertFalse(Main.validateCityStateZip("Dallas, ZZ 75201"));  // "ZZ" is not a valid state
   }

   @Test
   public void testInvalidStateNumeric() {
       assertFalse(Main.validateCityStateZip("Austin, 12 73301"));  // State cannot be numbers
   }

   @Test
   public void testInvalidStateTooLong() {
       assertFalse(Main.validateCityStateZip("Las Vegas, NEV 89109"));  // State abbreviation must be 2 letters
   }

   @Test
   public void testInvalidCityWithSpecialCharacters() {
       assertFalse(Main.validateCityStateZip("New@York, NY 10001"));  // City contains '@' which is invalid
   }

   @Test
   public void testValidStateEdgeCase() {
       assertTrue(Main.validateCityStateZip("Honolulu, HI 96813")); // Valid state abbreviation (edge case)
   }

   @Test
   public void testValidAddressWithAlternateZIP4Format() {
       assertTrue(Main.validateCityStateZip("Philadelphia, PA 19103-4567")); // ZIP+4 with different format
   }

   @Test
   public void testValidCityStateZipWithSpecialCharacterCity() {
       assertTrue(Main.validateCityStateZip("St. Louis, MO 63101")); // City with a period
```

```
    }

    @Test
    public void testValidCityStateZipWithApostropheCity() {
        assertTrue(Main.validateCityStateZip("O'Fallon, IL 62269")); // City with an apostrophe
    }

}
```

Military time (no colons used and leading 0 is included for times under 10)

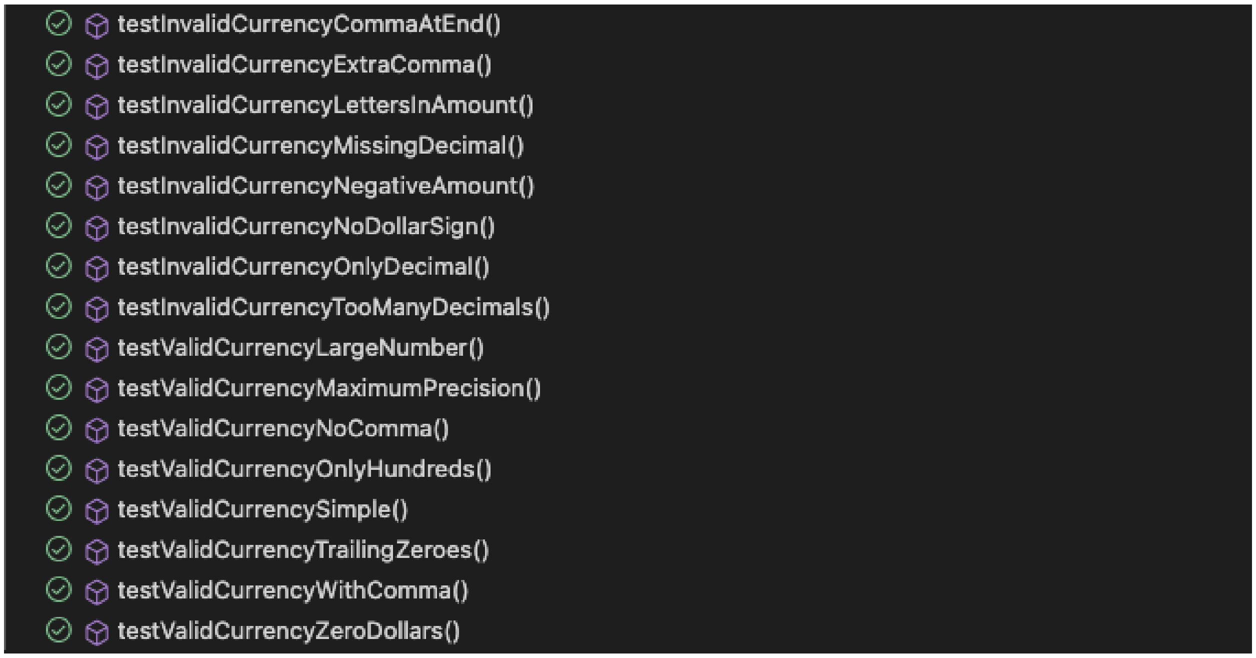


```
testInvalidMilitaryTimeContainsLetters()
testInvalidMilitaryTimeLeadingSpace()
testInvalidMilitaryTimeOutOfRangeHours()
testInvalidMilitaryTimeOutOfRangeMinutes()
testInvalidMilitaryTimeSpecialCharacters()
testInvalidMilitaryTimeTooLong()
testInvalidMilitaryTimeTooShort()
testInvalidMilitaryTimeTrailingSpace()
testValidMilitaryTimeAfternoon()
testValidMilitaryTimeEarlyMorning()
testValidMilitaryTimeEdgeCase()
testValidMilitaryTimeEvening()
testValidMilitaryTimeLateNight()
testValidMilitaryTimeLeadingZeroHour()
testValidMilitaryTimeMidmorning()
testValidMilitaryTimeMidnight()
testValidMilitaryTimeNoon()
```

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

public class MilitaryTimeValidatorTest {

    @Test
    public void testValidMilitaryTimeMidnight() {
        assertTrue(Main.validateMilitaryTime("0000")); // Midnight (start of day)
    }

    @Test
    public void testValidMilitaryTimeEarlyMorning() {
        assertTrue(Main.validateMilitaryTime("0930")); // Morning time
    }

    @Test
    public void testValidMilitaryTimeNoon() {
        assertTrue(Main.validateMilitaryTime("1200")); // Noon
    }
    @Test
    public void testValidMilitaryTimeMidmorning() {
        assertTrue(Main.validateMilitaryTime("1230")); // Midmorning time
    }

    @Test
    public void testValidMilitaryTimeAfternoon() {
        assertTrue(Main.validateMilitaryTime("1545")); // Afternoon time
    }

    @Test
    public void testValidMilitaryTimeEvening() {
        assertTrue(Main.validateMilitaryTime("2030")); // Evening time
    }

    @Test
    public void testValidMilitaryTimeLateNight() {
        assertTrue(Main.validateMilitaryTime("2359")); // Last valid minute before
midnight
    }

    @Test
    public void testValidMilitaryTimeLeadingZeroHour() {
        assertTrue(Main.validateMilitaryTime("0215")); // Leading zero is required for
hours < 10
    }

    @Test
    public void testValidMilitaryTimeEdgeCase() {
        assertTrue(Main.validateMilitaryTime("1959")); // Edge case before next hour
    }

    @Test
```

```java
    public void testInvalidMilitaryTimeOutOfRangeHours() {
        assertFalse(Main.validateMilitaryTime("2500")); // 25 is not a valid hour
    }

    @Test
    public void testInvalidMilitaryTimeOutOfRangeMinutes() {
        assertFalse(Main.validateMilitaryTime("1260")); // 60 is not a valid minute
    }

    @Test
    public void testInvalidMilitaryTimeTooShort() {
        assertFalse(Main.validateMilitaryTime("930")); // Missing leading zero
    }

    @Test
    public void testInvalidMilitaryTimeTooLong() {
        assertFalse(Main.validateMilitaryTime("12345")); // Extra digit
    }

    @Test
    public void testInvalidMilitaryTimeContainsLetters() {
        assertFalse(Main.validateMilitaryTime("12A0")); // Alphabetic character
    }

    @Test
    public void testInvalidMilitaryTimeSpecialCharacters() {
        assertFalse(Main.validateMilitaryTime("12:30")); // Contains colon
    }

    @Test
    public void testInvalidMilitaryTimeLeadingSpace() {
        assertFalse(Main.validateMilitaryTime(" 0930")); // Leading space
    }

    @Test
    public void testInvalidMilitaryTimeTrailingSpace() {
        assertFalse(Main.validateMilitaryTime("0930 ")); // Trailing space
    }

}
```

US Currency down to the penny (ex: $123,456,789.23)



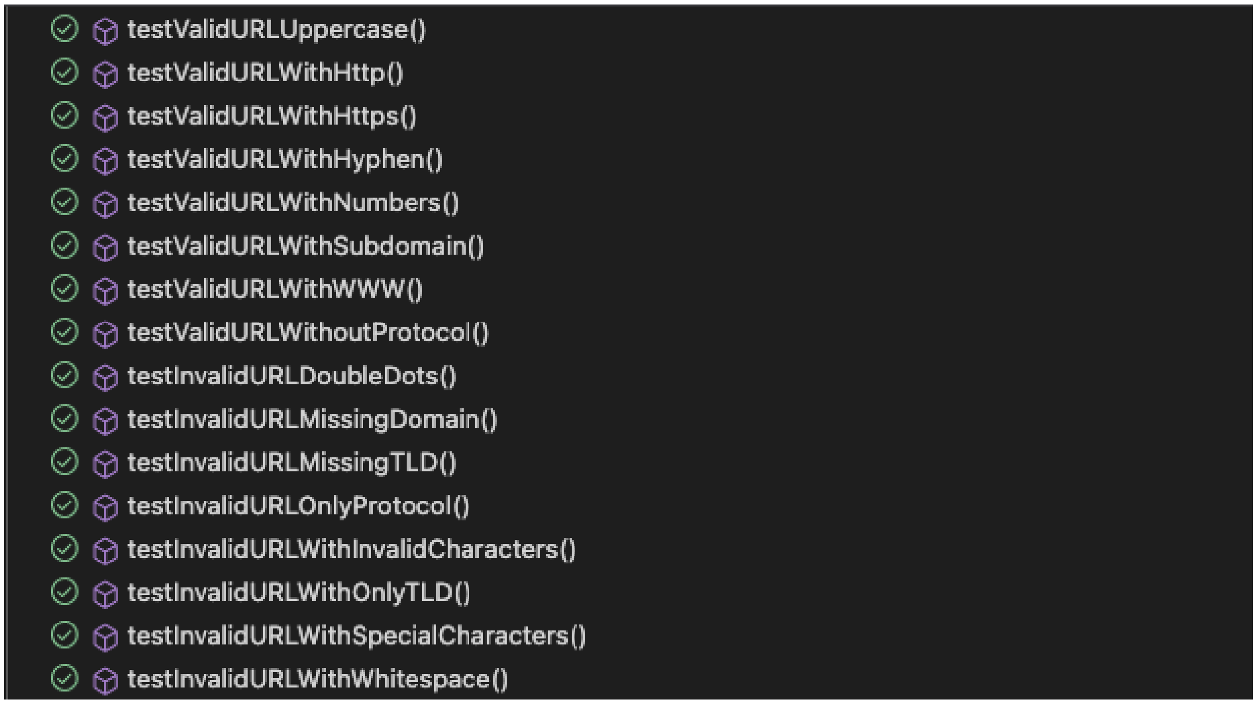
```
✓ testInvalidCurrencyCommaAtEnd()
✓ testInvalidCurrencyExtraComma()
✓ testInvalidCurrencyLettersInAmount()
✓ testInvalidCurrencyMissingDecimal()
✓ testInvalidCurrencyNegativeAmount()
✓ testInvalidCurrencyNoDollarSign()
✓ testInvalidCurrencyOnlyDecimal()
✓ testInvalidCurrencyTooManyDecimals()
✓ testValidCurrencyLargeNumber()
✓ testValidCurrencyMaximumPrecision()
✓ testValidCurrencyNoComma()
✓ testValidCurrencyOnlyHundreds()
✓ testValidCurrencySimple()
✓ testValidCurrencyTrailingZeroes()
✓ testValidCurrencyWithComma()
✓ testValidCurrencyZeroDollars()
```

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

import org.junit.*;

public class CurrencyValidatorTest {

   @Test
   public void testValidCurrencySimple() {
       assertTrue(Main.validateCurrency("$123.45"));
   }

   @Test
   public void testValidCurrencyNoComma() {
       assertTrue(Main.validateCurrency("$1000.99"));
   }

   @Test
   public void testValidCurrencyWithComma() {
       assertTrue(Main.validateCurrency("$1,234.56"));
   }

   @Test
```

```
public void testValidCurrencyLargeNumber() {
    assertTrue(Main.validateCurrency("$123,456,789.23"));
}

@Test
public void testValidCurrencyZeroDollars() {
    assertTrue(Main.validateCurrency("$0.99"));
}

@Test
public void testValidCurrencyOnlyHundreds() {
    assertTrue(Main.validateCurrency("$999.00"));
}

@Test
public void testValidCurrencyTrailingZeroes() {
    assertTrue(Main.validateCurrency("$5,678.90"));
}

@Test
public void testValidCurrencyMaximumPrecision() {
    assertTrue(Main.validateCurrency("$999,999,999.99"));
}

@Test
public void testInvalidCurrencyNoDollarSign() {
    assertFalse(Main.validateCurrency("123.45"));  // Missing $
}

@Test
public void testInvalidCurrencyMissingDecimal() {
    assertFalse(Main.validateCurrency("$123"));  // No .xx
}

@Test
public void testInvalidCurrencyTooManyDecimals() {
    assertFalse(Main.validateCurrency("$12.345"));  // More than two decimal places
}

@Test
public void testInvalidCurrencyExtraComma() {
    assertFalse(Main.validateCurrency("$1,,234.56"));  // Extra comma
```

```java
    }

    @Test
    public void testInvalidCurrencyOnlyDecimal() {
        assertFalse(Main.validateCurrency("$.99"));  // No whole number part
    }

    @Test
    public void testInvalidCurrencyCommaAtEnd() {
        assertFalse(Main.validateCurrency("$1,234,"));  // Ends in a comma
    }

    @Test
    public void testInvalidCurrencyLettersInAmount() {
        assertFalse(Main.validateCurrency("$12a,456.78"));  // Contains letter
    }

    @Test
    public void testInvalidCurrencyNegativeAmount() {
        assertFalse(Main.validateCurrency("-$123.45"));  // Negative sign before $
    }
}
```

URL, optionally including http:// or https://, upper and lower case should be accepted



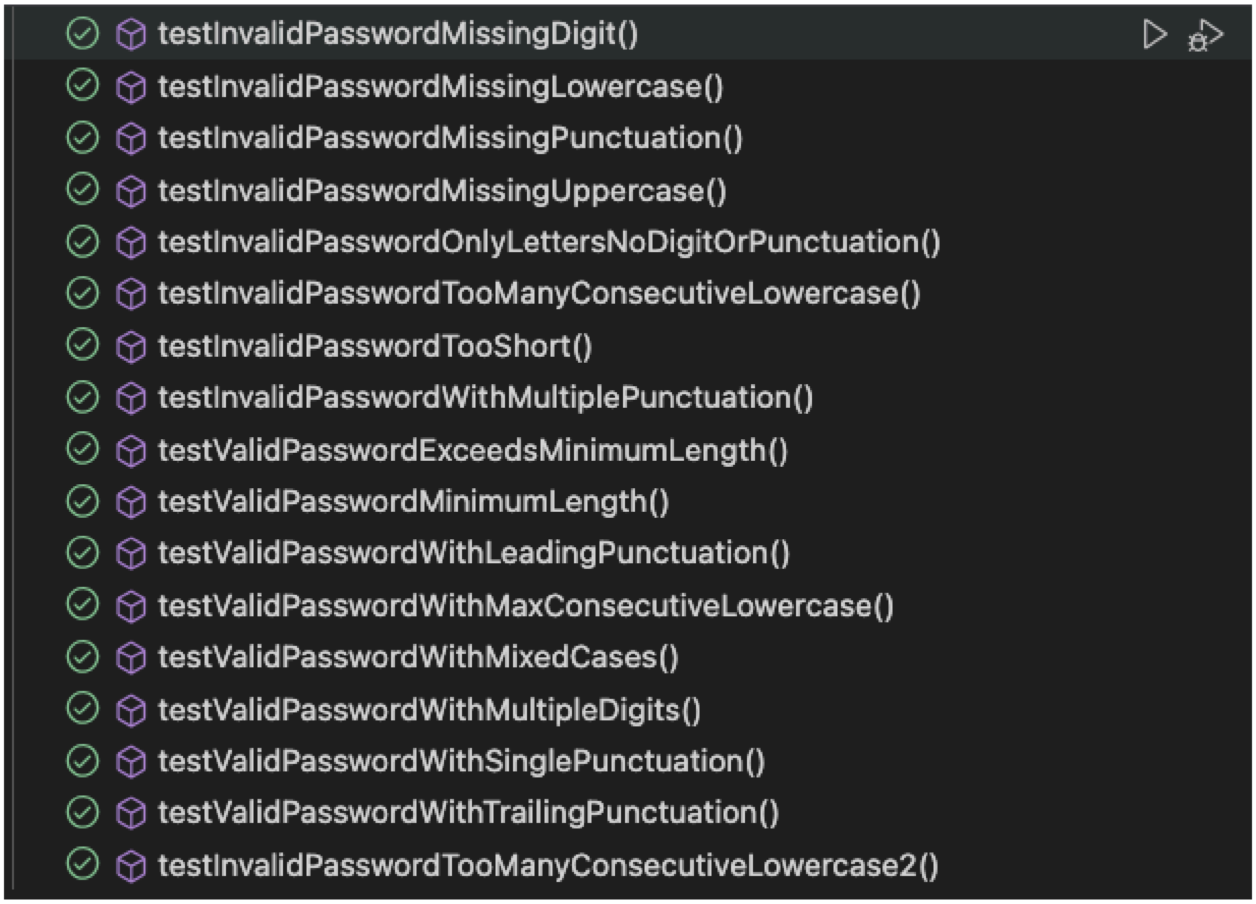
```
✓ testValidURLUppercase()
✓ testValidURLWithHttp()
✓ testValidURLWithHttps()
✓ testValidURLWithHyphen()
✓ testValidURLWithNumbers()
✓ testValidURLWithSubdomain()
✓ testValidURLWithWWW()
✓ testValidURLWithoutProtocol()
✓ testInvalidURLDoubleDots()
✓ testInvalidURLMissingDomain()
✓ testInvalidURLMissingTLD()
✓ testInvalidURLOnlyProtocol()
✓ testInvalidURLWithInvalidCharacters()
✓ testInvalidURLWithOnlyTLD()
✓ testInvalidURLWithSpecialCharacters()
✓ testInvalidURLWithWhitespace()
```

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

public class URLValidatorTest {

   @Test
   public void testValidURLWithHttp() {
       assertTrue(Main.validateURL("http://example.com")); // Basic HTTP URL
   }

   @Test
   public void testValidURLWithHttps() {
       assertTrue(Main.validateURL("https://example.com")); // Basic HTTPS URL
   }

   @Test
   public void testValidURLWithoutProtocol() {
       assertTrue(Main.validateURL("example.com")); // No http/https, still valid

   }

   @Test
   public void testValidURLWithSubdomain() {
       assertTrue(Main.validateURL("https://sub.example.com")); // Valid subdomain
   }

   @Test
   public void testValidURLUppercase() {
       assertTrue(Main.validateURL("HTTPS://EXAMPLE.COM")); // Uppercase letters
should be valid
   }

   @Test
   public void testValidURLWithWWW() {
       assertTrue(Main.validateURL("www.example.com")); // "www" should be valid
   }

   @Test
   public void testValidURLWithHyphen() {
       assertTrue(Main.validateURL("https://my-site.com")); // Hyphen in domain name
   }
```

```java
    @Test
    public void testValidURLWithNumbers() {
        assertTrue(Main.validateURL("http://123example.com")); // Numbers in domain are
valid
    }

    @Test
    public void testInvalidURLMissingDomain() {
        assertFalse(Main.validateURL("http://")); // No domain name
    }

    @Test
    public void testInvalidURLOnlyProtocol() {
        assertFalse(Main.validateURL("https://")); // Just protocol, no domain
    }

    @Test
    public void testInvalidURLWithInvalidCharacters() {
        assertFalse(Main.validateURL("http://exa*mple.com")); // Invalid character '*'
    }

    @Test
    public void testInvalidURLDoubleDots() {
        assertFalse(Main.validateURL("https://example..com")); // Double dots in domain
    }

    @Test
    public void testInvalidURLMissingTLD() {
        assertFalse(Main.validateURL("http://example")); // No .com, .org, etc.
    }

    @Test
    public void testInvalidURLWithWhitespace() {
        assertFalse(Main.validateURL("http://example .com")); // Space in domain
    }

    @Test
    public void testInvalidURLWithSpecialCharacters() {
        assertFalse(Main.validateURL("https://example@.com")); // '@' is not valid in
domain
    }
```

```
   @Test
   public void testInvalidURLWithOnlyTLD() {
       assertFalse(Main.validateURL(".com")); // Missing domain name before TLD
   }
}
```

A password that contains at least 10 characters and includes at least one upper case character, one lower case character, one digit, one punctuation mark, and does not have more than 3 consecutive lower case characters



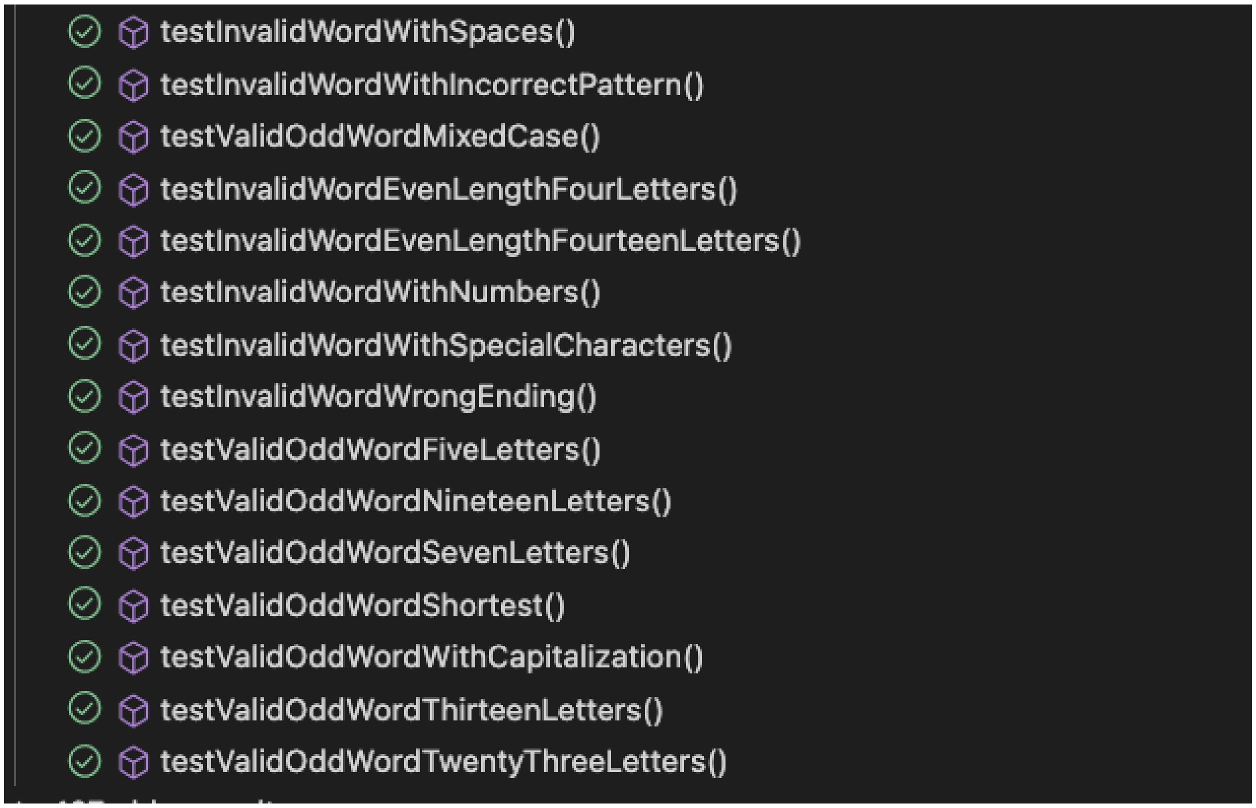
```
✓ testInvalidPasswordMissingDigit()
✓ testInvalidPasswordMissingLowercase()
✓ testInvalidPasswordMissingPunctuation()
✓ testInvalidPasswordMissingUppercase()
✓ testInvalidPasswordOnlyLettersNoDigitOrPunctuation()
✓ testInvalidPasswordTooManyConsecutiveLowercase()
✓ testInvalidPasswordTooShort()
✓ testInvalidPasswordWithMultiplePunctuation()
✓ testValidPasswordExceedsMinimumLength()
✓ testValidPasswordMinimumLength()
✓ testValidPasswordWithLeadingPunctuation()
✓ testValidPasswordWithMaxConsecutiveLowercase()
✓ testValidPasswordWithMixedCases()
✓ testValidPasswordWithMultipleDigits()
✓ testValidPasswordWithSinglePunctuation()
✓ testValidPasswordWithTrailingPunctuation()
✓ testInvalidPasswordTooManyConsecutiveLowercase2()
```

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

public class PassWordValidationTest {

   @Test
```

```
    public void testValidPasswordMinimumLength() {
        assertTrue(Main.validatePassword("Abc!123456")); // Exactly 10 characters,
meets all conditions
    }

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

    @Test
    public void testValidPasswordWithMaxConsecutiveLowercase() {
        assertTrue(Main.validatePassword("Aabc!123DEF")); // 3 consecutive lowercase
allowed
    }

    @Test
    public void testValidPasswordWithMixedCases() {
        assertTrue(Main.validatePassword("Pass1!WordX")); // Proper mix of upper,
lower, digit, punctuation
    }

    @Test
    public void testValidPasswordWithSinglePunctuation() {
        assertTrue(Main.validatePassword("Secure!Pass12")); // Uses exactly one
punctuation mark
    }

    @Test
    public void testValidPasswordWithLeadingPunctuation() {
        assertTrue(Main.validatePassword("!Pass12WordX")); // Punctuation at start
    }

    @Test
    public void testValidPasswordWithTrailingPunctuation() {
        assertTrue(Main.validatePassword("Pass12WordX!")); // Punctuation at end
    }

    @Test
    public void testValidPasswordWithMultipleDigits() {
        assertTrue(Main.validatePassword("Strong!9876Abc")); // Extra numbers but still
valid
    }


    @Test
    public void testInvalidPasswordTooShort() {
        assertFalse(Main.validatePassword("Ab1!cde")); // Less than 10 characters
    }

    @Test
    public void testInvalidPasswordMissingUppercase() {
        assertFalse(Main.validatePassword("abcdef!1234")); // No uppercase letter
    }

    @Test
    public void testInvalidPasswordMissingLowercase() {
        assertFalse(Main.validatePassword("ABCDEF!1234")); // No lowercase letter
    }

    @Test
    public void testInvalidPasswordMissingDigit() {
        assertFalse(Main.validatePassword("Abcdefgh!X")); // No digit
    }

    @Test
    public void testInvalidPasswordMissingPunctuation() {
        assertFalse(Main.validatePassword("Abcdef12345")); // No punctuation
    }

    @Test
    public void testInvalidPasswordTooManyConsecutiveLowercase() {
        assertFalse(Main.validatePassword("Abcdeeee!123")); // More than 3 consecutive
lowercase letters
    }
    @Test
    public void testInvalidPasswordTooManyConsecutiveLowercase2() {
        assertFalse(Main.validatePassword("Abcdaerfcsfgrezvvfefergverfrfrrrfrf!123"));
// More than 3 consecutive lowercase letters
    }
```

```java
    @Test
    public void testInvalidPasswordWithMultiplePunctuation() {
        assertFalse(Main.validatePassword("Secure!Pass@12")); // More than one
punctuation mark
    }

    @Test
    public void testInvalidPasswordOnlyLettersNoDigitOrPunctuation() {
        assertFalse(Main.validatePassword("Abcdefghijkl")); // No digit, no punctuation
    }
}
```

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



```
✓ testInvalidWordWithSpaces()
✓ testInvalidWordWithIncorrectPattern()
✓ testValidOddWordMixedCase()
✓ testInvalidWordEvenLengthFourLetters()
✓ testInvalidWordEvenLengthFourteenLetters()
✓ testInvalidWordWithNumbers()
✓ testInvalidWordWithSpecialCharacters()
✓ testInvalidWordWrongEnding()
✓ testValidOddWordFiveLetters()
✓ testValidOddWordNineteenLetters()
✓ testValidOddWordSevenLetters()
✓ testValidOddWordShortest()
✓ testValidOddWordWithCapitalization()
✓ testValidOddWordThirteenLetters()
✓ testValidOddWordTwentyThreeLetters()
```

```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(Main.validateOddWord("ion")); // 3 total letters (odd)
   }

   @Test
   public void testValidOddWordFiveLetters() {
       assertTrue(Main.validateOddWord("union")); // 5 total letters (odd)
   }

   @Test
   public void testValidOddWordSevenLetters() {
       assertTrue(Main.validateOddWord("invention")); // 7 total letters (odd)
   }

   @Test
   public void testValidOddWordMixedCase() {
       assertTrue(Main.validateOddWord("Suppression")); // Case insensitive check
   }
   @Test
   public void testValidOddWordThirteenLetters() {
       assertTrue(Main.validateOddWord("participation")); // 13 total letters (odd)
   }

   @Test
   public void testValidOddWordNineteenLetters() {
       assertTrue(Main.validateOddWord("conceptualization")); // 19 total letters
(odd)
   }

   @Test
   public void testValidOddWordTwentyThreeLetters() {
       assertTrue(Main.validateOddWord("Overintellectualization")); // 22 total
letters (odd)
   }

   @Test
   public void testValidOddWordWithCapitalization() {
       assertTrue(Main.validateOddWord("Commercialization")); // Should be case
insensitive
   }

   @Test
```

```
    public void testInvalidWordWrongEnding() {
        assertFalse(Main.validateOddWord("motivation")); // not correct length
    }

    @Test
    public void testInvalidWordWithNumbers() {
        assertFalse(Main.validateOddWord("m0tion")); // Contains a number
    }

    @Test
    public void testInvalidWordWithSpecialCharacters() {
        assertFalse(Main.validateOddWord("moti@nion")); // Contains special character
    }

    @Test
    public void testInvalidWordWithSpaces() {
        assertFalse(Main.validateOddWord(" expulsion ")); // Leading/trailing spaces
    }

    @Test
    public void testInvalidWordWithIncorrectPattern() {
        assertFalse(Main.validateOddWord("bahion")); // Doesn't follow the regex
pattern
    }

    @Test
    public void testInvalidWordEvenLengthFourLetters() {
        assertFalse(Main.validateOddWord("xion")); // "xion" has 4 letters (even)
    }

    @Test
    public void testInvalidWordEvenLengthFourteenLetters() {
        assertFalse(Main.validateOddWord("disintegration")); // 14 total letters (even)
    }
}
```