Coverage for **validador_test.py**: 100%

38 statements | 38 run | 0 missing | 1 excluded


```
1  """
2  Módulo de testes para o validador de CPF.
3  """
4
5  import unittest
6  from validador_cpf import LimpezaCPF, ValidadorCPF
7
8
9  class TestLimpezaCPF(unittest.TestCase):
10     """
11     Testes para a classe LimpezaCPF.
12     """
13
14     def test_limpeza(self):
15         """
16         DADO cpf COM caracteres não numéricos
17         QUANDO o método limpeza é chamado
18         ENTÃO o cpf é retornado sem os caracteres não numéricos
19         """
20         self.assertEqual(LimpezaCPF.limpeza("123.456.789-00"), "12345678900")
21         self.assertEqual(LimpezaCPF.limpeza("000.111.222-33"), "00011122233")
22         self.assertEqual(LimpezaCPF.limpeza("abc123def456"), "123456")
23
24
25 class TestValidadorCPF(unittest.TestCase):
26     """
27     Testes para a classe ValidadorCPF.
28     """
29
30     def test_calcular_digito(self):
31         """
32         DADO uma sequência de números e a posição do dígito
33         QUANDO o método _calcular_digito é chamado
34         ENTÃO o dígito é calculado corretamente
35         """
36         self.assertEqual(ValidadorCPF._calcular_digito("987654321", 10), 0)
37         self.assertEqual(ValidadorCPF._calcular_digito("9876543210", 11), 10)
38         self.assertEqual(ValidadorCPF._calcular_digito("787159503", 10), 2)
39         self.assertEqual(ValidadorCPF._calcular_digito("7871595032", 11), 8)
40
41     def test_obter_digito_verificador(self):
42         """
43         DADO uma sequência de números e a posição do dígito
44         QUANDO o método _obter_digito_verificador é chamado
45         ENTÃO o dígito verificador é obtido corretamente
46         """
47         self.assertEqual(
48             ValidadorCPF._obter_digito_verificador("987654321", 10), 0)
```

```
        self.assertEqual(
            ValidadorCPF._obter_digito_verificador("9876543210", 11), 0)
        self.assertEqual(
            ValidadorCPF._obter_digito_verificador("787159503", 10), 2)
        self.assertEqual(
            ValidadorCPF._obter_digito_verificador("7871595032", 11), 8)

    def test_validar_digitos(self):
        """
        DADO uma sequência de números com dígitos verificadores
        QUANDO o método _validar_digitos é chamado
        ENTÃO os dígitos verificadores são validados corretamente
        """
        self.assertTrue(ValidadorCPF._validar_digitos("98765432100"))
        self.assertTrue(ValidadorCPF._validar_digitos("78715950328"))
        self.assertFalse(ValidadorCPF._validar_digitos("00000000000"))
        self.assertFalse(ValidadorCPF._validar_digitos("498703"))

    def test_validacao_cpf_valido(self):
        """
        DADO um CPF válido
        QUANDO o método validacao_cpf é chamado
        ENTÃO retorna True
        """
        self.assertTrue(ValidadorCPF("123.456.789-09").validacao_cpf())
        self.assertTrue(ValidadorCPF("98765432100").validacao_cpf())
        self.assertTrue(ValidadorCPF("235.998.567-10").validacao_cpf())

    def test_validacao_cpf_invalido(self):
        """
        DADO um CPF inválido
        QUANDO o método validacao_cpf é chamado
        ENTÃO retorna False
        """
        self.assertFalse(ValidadorCPF("11111111111").validacao_cpf())
        self.assertFalse(ValidadorCPF("123.456.789-00").validacao_cpf())
        self.assertFalse(ValidadorCPF("15698192022").validacao_cpf())
        self.assertFalse(ValidadorCPF("005.998.567-11").validacao_cpf())

    def test_validacao_cpf_formato_invalido(self):
        """
        DADO um CPF com formato inválido
        QUANDO o método validacao_cpf é chamado
        ENTÃO retorna False
        """
        self.assertFalse(ValidadorCPF("123.456.789").validacao_cpf())
        self.assertFalse(ValidadorCPF("abc123def456").validacao_cpf())
        self.assertFalse(ValidadorCPF("123").validacao_cpf())
        self.assertFalse(ValidadorCPF("").validacao_cpf())


if __name__ == '__main__':
    unittest.main()  # pragma: no cover
```