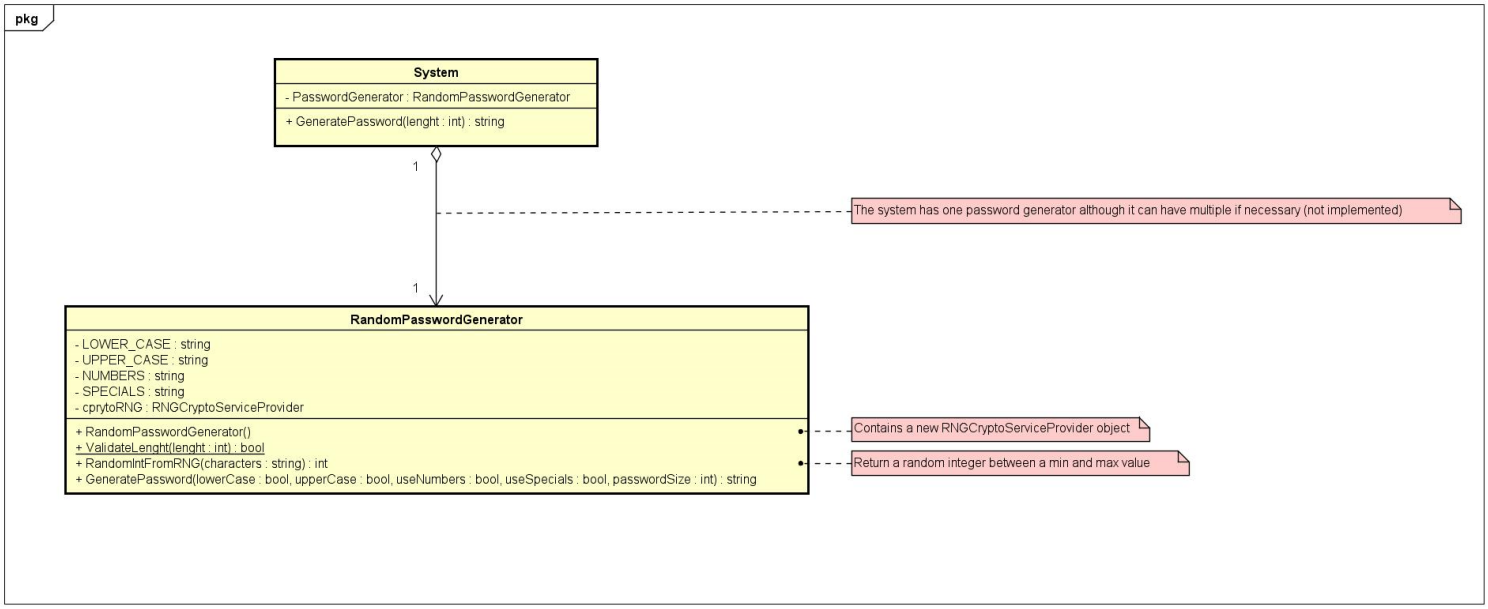
Random Password Generator

Cecilia Belon

1. UML Diagram



2. Code

- RandomPasswordGenerator

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Security.Cryptography;

namespace Domain
{
    public class RandomPasswordGenerator
    {
        #region attributes
        const string LOWER_CASE = "abcdefghijklmnopqursuvwxyz";
        const string UPPER_CAES = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        const string NUMBERS = "123456789";
        const string SPECIALS = "!@-$%+*()#/";
        private RNGCryptoServiceProvider CprytoRNG { get; set; }
        #endregion

        #region constructor
        /// <summary>
```

```
        /// Constructor that generates a new password generator with a new
CprytoRNG object
        /// </summary>
        public RandomPasswordGenerator()
        {
            CprytoRNG = new RNGCryptoServiceProvider();
        }
        #endregion

        #region methods
        /// <summary>
        /// Validates lenght is greater or equals to ten
        /// </summary>
        /// <param name="lenght"></param>
        /// <returns></returns>
        public static bool ValidateLenght(int lenght)
        {
            return lenght >= 10;
        }

        /// <summary>
        /// Return a random integer between a min and max value.
        /// </summary>
        /// <param name="characters"></param>
        /// <returns></returns>
        public int RandomIntFromRNG(string characters)
        {
            int min = 0;
            int max = characters.Length;

            // Generate four random bytes
            byte[] four_bytes = new byte[4];
            CprytoRNG.GetBytes(four_bytes);

            // Convert the bytes to a UInt32
            UInt32 scale = BitConverter.ToUInt32(four_bytes, 0);

            // And use that to pick a random number >= min and < max
            return (int)(min + (max - min) * (scale / (uint.MaxValue + 1.0)));
        }
        /// <summary>
        ///
        /// </summary>
        /// <param name="useLowercase"></param>
        /// <param name="useUppercase"></param>
        /// <param name="useNumbers"></param>
        /// <param name="useSpecial"></param>
        /// <param name="passwordSize"></param>
        /// <returns></returns>
        public string GeneratePassword(bool useLowercase, bool useUppercase, bool
useNumbers, bool useSpecial,
            int passwordSize)
        {
            char[] _password = new char[passwordSize];
            string charSet = ""; // Initialise to blank
            int counter;

            // Build up the character set to choose from
            if (useLowercase) charSet += LOWER_CASE;

            if (useUppercase) charSet += UPPER_CAES;
```

```
            if (useNumbers) charSet += NUMBERS;

            if (useSpecial) charSet += SPECIALS;

            for (counter = 0; counter < passwordSize; counter++)
            {
                _password[counter] = charSet[RandomIntFromRNG(charSet)];
            }

            return String.Join(null, _password);
        }
        #endregion
    }
}
```

- Sistema

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Domain
{
    public class Sistema
    {
        #region attributes
        public RandomPasswordGenerator PasswordGenerator { get; set; }
        #endregion

        #region constructor
        public Sistema()
        {
            PasswordGenerator = new RandomPasswordGenerator();
        }
        #endregion

        #region methods
        /// <summary>
        /// According to lenght of password generates a new password or gives an
error message.
        /// </summary>
        /// <param name="lenght"></param>
        /// <returns></returns>
        public string GeneratePassword(int lenght=10)
        {
            string password = "Lenght must be greater or equals to 10.";

            if (RandomPasswordGenerator.ValidateLenght(lenght))
            {
                password = PasswordGenerator.GeneratePassword(true, true, true,
true, lenght);
            }

            return password;
        }
        #endregion
```

```
    }
}
```