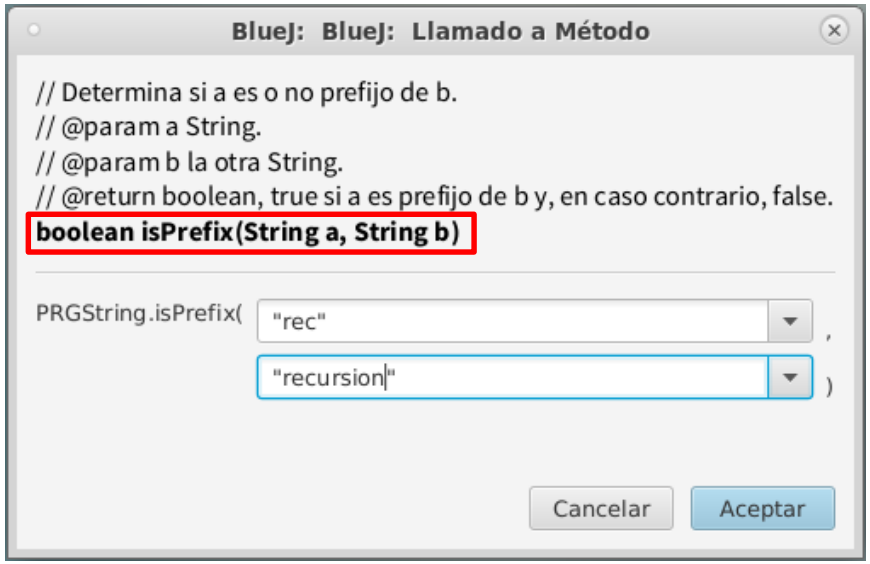
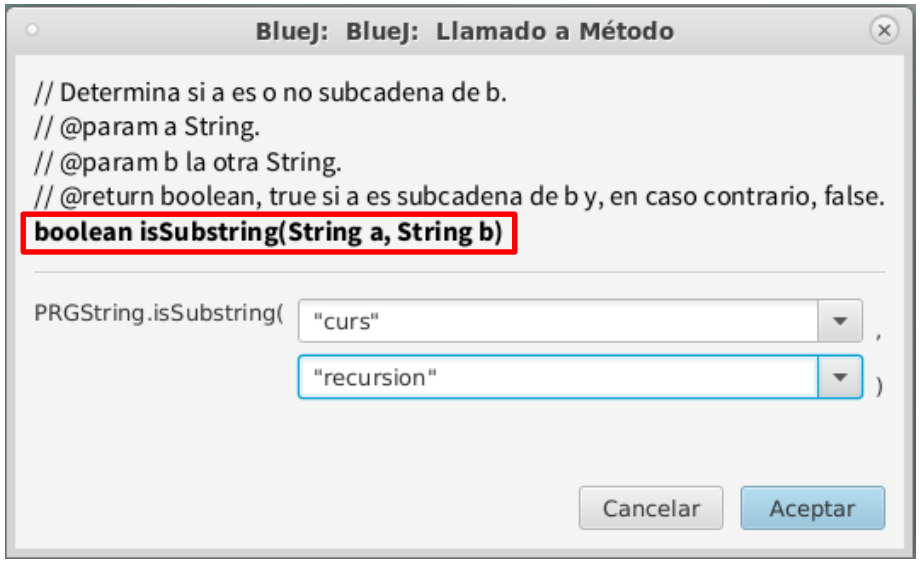
PRG – Práctica 2

Resolución de algunos problemas con recursión

Marisa Llorens (mllorens@dsic.upv.es)





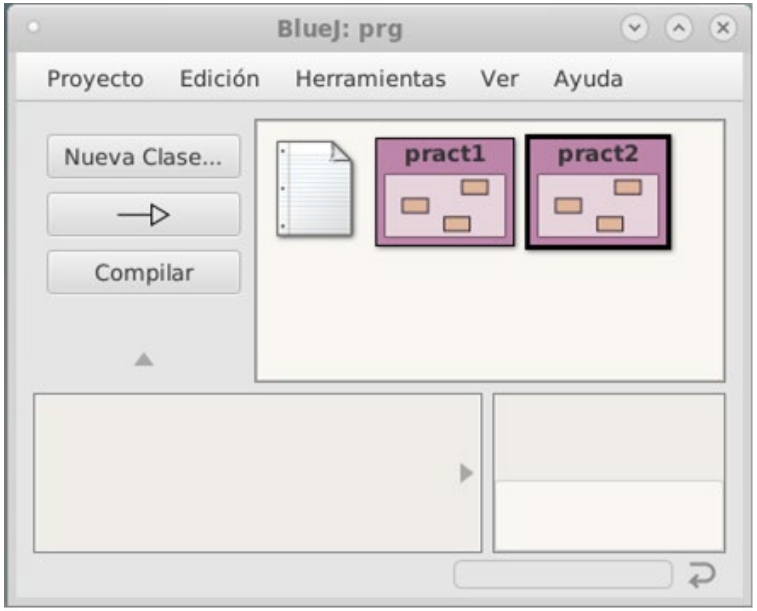
1. Contexto y trabajo previo

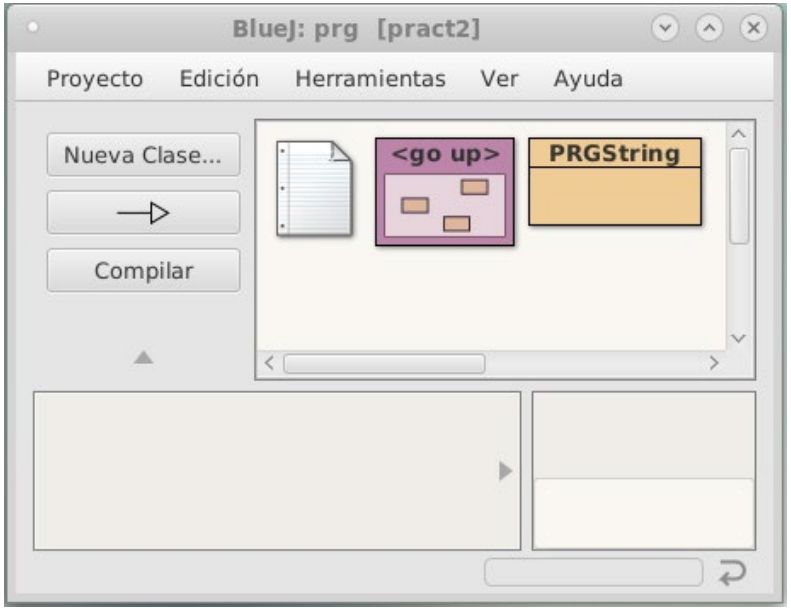
En esta práctica se propone la resolución de forma recursiva de dos problemas con `Strings`. Para ello, se diseñarán los métodos correspondientes y las clases de prueba para asegurar que las soluciones de los problemas sean correctas.

Es conveniente haber estudiado la sección 10.6 *Recursividad con objetos de tipo* ***`String`*** de la 3ª edición del libro de la asignatura[1] y haber comprendido algunos ejemplos como el problema de contar el número de caracteres `'a'` en cierta `String s`.

Actividad 1: Creación del paquete BlueJ `pract2`

Abre el proyecto *BlueJ* de trabajo de la asignatura (`prg`) y crea un nuevo paquete `pract2`. Agrega al paquete el fichero `PRGString.java` que habrás descargado previamente de la carpeta `Recursos/Laboratorio/Práctica 2` de la PoliformaT de PRG. La clase `PRGString` es una clase de utilidades que incluye los métodos que resuelven el problema de contar el número de `'a'`s en una `String s` (sección 10.6 del libro) y los métodos (a completar) que resuelven los problemas que se te plantean a continuación.





https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/String.html

Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods | Deprecated Methods

Modifier and Type	Method	Description
char	**charAt**(int index)	Returns the char value at the specified index.
int	**length**()	Returns the length of this string.
String	**substring**(int beginIndex)	Returns a string that is a substring of this string.
String	**substring**(int beginIndex, int endIndex)	Returns a string that is a substring of this string.

substring

```
public String substring(int beginIndex)
```

Returns a string that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string.

Examples:

```
"unhappy".substring(2) returns "happy"
"Harbison".substring(3) returns "bison"
"emptiness".substring(9) returns "" (an empty string)
```

Parameters:

beginIndex - the beginning index, inclusive.

Returns:

the specified substring.

Throws:

IndexOutOfBoundsException - if beginIndex is negative or larger tha

substring

```
public String substring(int beginIndex, int endIndex)
```

Returns a string that is a substring of this string. The substring begins at the specified beginIndex and extends to the character at index endIndex - 1. Thus the length of the substring is endIndex-beginIndex.

Examples:

```
"hamburger".substring(4, 8) returns "urge"
"smiles".substring(1, 5) returns "mile"
```

Parameters:

beginIndex - the beginning index, inclusive.

endIndex - the ending index, exclusive.

Returns:

the specified substring.

Throws:

IndexOutOfBoundsException - if the beginIndex is negative, or endIndex is larger than the length of this String object, or beginIndex is larger than endIndex.

```java
package pract2;

/**
 * Clase PRGString: clase de utilidades con métodos para trabajar con Strings.
 *
 * @author PRG - ETSINF - DSIC - UPV
 * @version Curso 2019/2020
 */
public class PRGString {
    /** No hay objetos de esta clase. */
    private PRGString() { }

    /**
     * Devuelve el número de 'a's en la String dada.
     * @param s String en la que se quieren contar las 'a's.
     * @return int.
     */
    public static int countA(String s) {
        // Caso base: String vacía
        if (s.length() == 0) { return 0; }
        // Caso general: String no vacía. Tratar la substring posterior.
        else if (s.charAt(0) == 'a') { return 1 + countA(s.substring(1)); }
        else { return countA(s.substring(1)); }
    }

    /**
     * Devuelve el número de 'a's en la String dada.
     * @param s String en la que se quieren contar las 'a's.
     * @return int.
     */
    public static int countA2(String s) {
        // Caso base: String vacía
        if (s.length() == 0) { return 0; }
        // Caso general: String no vacía. Tratar la substring anterior.
        else if (s.charAt(s.length() - 1) == 'a') {
            return 1 + countA2(s.substring(0, s.length() - 1));
        } else { return countA2(s.substring(0, s.length() - 1)); }
    }
```

2. Problema A. *Prefijo*

Dadas dos `Strings` `a` y `b`, potencialmente vacías, se dice que `a` es *prefijo* de `b` cuando todos los caracteres de `a` están consecutivos, en el mismo orden original, al comienzo de `b`.

Consecuencia de la definición anterior es que la *cadena vacía* es prefijo de cualquier otra, incluso si esa otra también estuviese vacía. Nota, por otra parte, que una cadena no puede ser prefijo de otra si la primera es de longitud mayor que la segunda.

Actividad 2: método `isPrefix(String, String)`

Define recursivamente un método `isPrefix(String, String)` para comprobar si una cadena es prefijo de otra. Para ello:

- Establece los casos base y general de la recursión definiendo, además, la solución del problema en cada uno de dichos casos. La cabecera del método (en la que no hay parámetros posicionales) debe ser necesariamente la que sigue:

  ```
  public static boolean isPrefix(String a, String b)
  ```

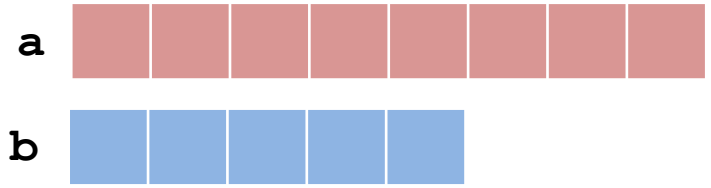
- Documenta adecuadamente el método, explicitando cuáles son sus parámetros, el tipo de su resultado y, caso de haberla, su precondición.
- Comprueba que el código del método sigue las normas de estilo usando el *Checkstyle* de *BlueJ* y corrígelo si no es el caso.

```java
/**
 * Determina si a es o no prefijo de b.
 * -- COMPLETAR --
 */
public static boolean isPrefix(String a, String b) {
    /* COMPLETAR */
    return true;
}
```

Análisis de casos

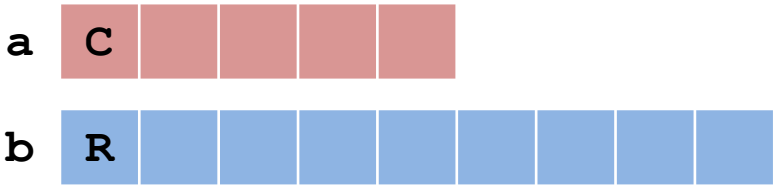
Caso base:

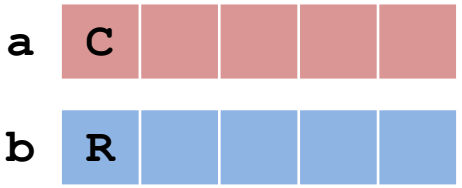
- La **`String a`** es vacía, **`a`** es **prefijo** de **`b`**. **`a`** `""`
- La **`String a`** es de longitud mayor que la **`String b`**, **`a`** **no** es **prefijo** de **`b`**.



Caso general o recursivo:

- La **`String a`** no es vacía ni de longitud mayor que la **`String b`**,
  - Si el primer carácter de **`a`** **no** coincide con el primer carácter de **`b`**, **`a`** **no** es **prefijo** de **`b`**.





  - Si el primer carácter de **`a`** **sí** coincide con el primero de **`b`**, entonces **`a`** es **prefijo** de **`b`** si la **substring** de **`a`** desde el carácter **1** en adelante es **prefijo** de la **substring** de **`b`** desde el carácter **1** en adelante. En caso contrario, **`a`** **no** es **prefijo** de **`b`**.

Actividad 3: validación del método `isPrefix(String, String)`

Escribe una clase programa `TestIsPrefix` que permita ejecutar el método con diferentes datos para comprobar que no hay errores de ejecución y que el resultado que se devuelve en cada caso es el correcto.

Los datos a probar deben reflejar las distintas situaciones que se pueden dar en la ejecución del método, tales como, por ejemplo: que ambas cadenas estén vacías, que lo esté solo una de ellas, que la primera cadena sea más larga que la segunda, que la primera cadena sea prefijo o no de la segunda, etc. En la tabla siguiente se detallan los diferentes casos, con instancias concretas y el resultado esperado para cada caso.

Caso	`a`	`b`	**Resultado**
`a` y `b` vacías	`""`	`""`	`true`
Solo `a` vacía	`""`	`"recursion"`	`true`
Solo `b` vacía	`"recursion"`	`""`	`false`
`a` de mayor longitud que `b`	`"recursion"`	`"rec"`	`false`
`a` y `b` de igual longitud y `a` es prefijo de `b`	`"recursion"`	`"recursion"`	`true`
`a` y `b` de igual longitud y `a` no es prefijo de `b`	`"123456789"`	`"recursion"`	`false`
`a` de menor longitud que `b` y `a` es prefijo de `b`	`"rec"`	`"recursion"`	`true`
`a` de menor longitud que `b` y `a` no es prefijo de `b`:			
- por el primer carácter	`"pecur"`	`"recursion"`	`false`
- por el último carácter	`"recurso"`	`"recursion"`	`false`
- por un carácter intermedio	`"remursi"`	`"recursion"`	`false`

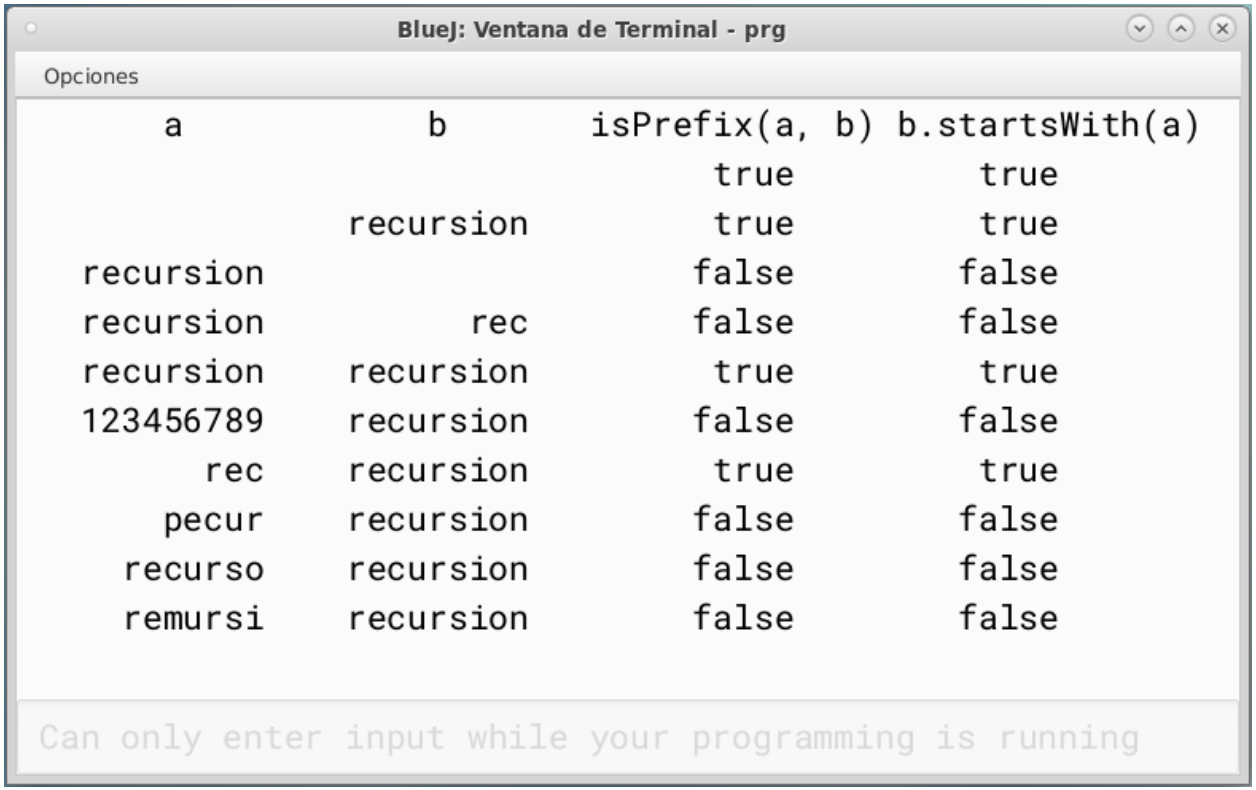
La clase `TestIsPrefix` debe incluir un método con el siguiente perfil:

```
private static void testIsPrefix(String a, String b)
```

que muestre por pantalla las `String`s de prueba, el resultado de tu método `isPrefix(String, String)` y el resultado esperado. Para esto último, puedes utilizar el método `startsWith(String)` de la clase `String`.

`boolean`	`startsWith(String prefix)`	Tests if this string starts with the specified prefix.

El `main` debe invocar al método `testIsPrefix(String, String)` para cada caso de prueba. Puedes definir un array de `String` para almacenar las diferentes instancias de los casos a probar.



```java
public class TestIsPrefix {
    /** No hay objetos de esta clase */
    private TestIsPrefix() { }

    public static void main(String[] args) {
        String[] s = {"", "rec", "pecur", "recurso", "remursi",
                      "123456789", "recursion"};

        System.out.printf("%8s %12s %20s %12s\n",
            "a", "b", "isPrefix(a, b)", "b.startsWith(a)");

        // a y b vacías
        testIsPrefix(s[0], s[0]);
        // solo a vacía

        // solo b vacía

        // a de mayor longitud que b

        // a y b de igual longitud y a es prefijo de b

        // a y b de igual longitud y a no es prefijo de b

        // a de menor longitud que b y a es prefijo de b

        // a de menor longitud que b y a no es prefijo de b:
        // por el primer carácter

        // a de menor longitud que b y a no es prefijo de b:
        // por el ultimo carácter

        // a de menor longitud que b y a no es prefijo de b:
        // por un carácter intermedio

    }
```

```java
private static void testIsPrefix(String a, String b) {

}
```

3. Problema B. *Subcadena*

Dadas dos `Strings` `a` y `b`, potencialmente vacías, se dice que `a` es *subcadena* de `b` cuando todos los caracteres de `a` están consecutivos, en el mismo orden original, en algún lugar de `b`. O, lo que es lo mismo, cuando `a` es prefijo de `b` o de alguna de las posibles subcadenas de `b`.

Naturalmente, igual que ocurría en el caso de `isPrefix(String, String)`, se puede ver que la *cadena vacía* es subcadena de cualquier otra, incluso si esa otra también estuviese vacía. Además, una cadena no puede ser subcadena de otra si la primera es de longitud mayor que la segunda.

Actividad 4: método `isSubstring(String, String)`

Define recursivamente, **en términos de** `isPrefix(String, String)`, el método `isSubstring(String)`, para poder comprobar si una cadena es subcadena de otra. Para ello:

- Enuncia los casos base y general de la recursión, definiendo la solución del problema en cada caso. La cabecera del método deberá ser necesariamente:

  ```
  public static boolean isSubstring(String a, String b)
  ```

  Nota que, al igual que para la operación `isPrefix(String, String)`, no hay parámetros posicionales en la cabecera anterior.

- Documenta adecuadamente el método, explicitando sus parámetros y resultado así como, caso de haberla, su precondición.

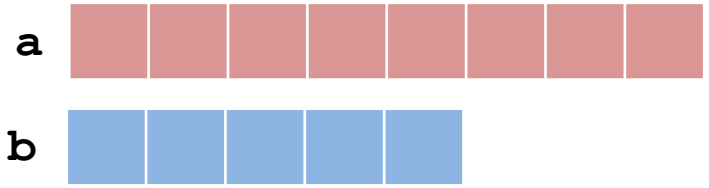
- Comprueba que el código del método sigue las normas de estilo usando el *Checkstyle* de *BlueJ* y corrígelo si no es el caso.

```java
/**
 * Determina si a es o no subcadena de b.
 * -- COMPLETAR --
 */
public static boolean isSubstring(String a, String b) {
    /* COMPLETAR */
    return true;
}
```

Análisis de casos

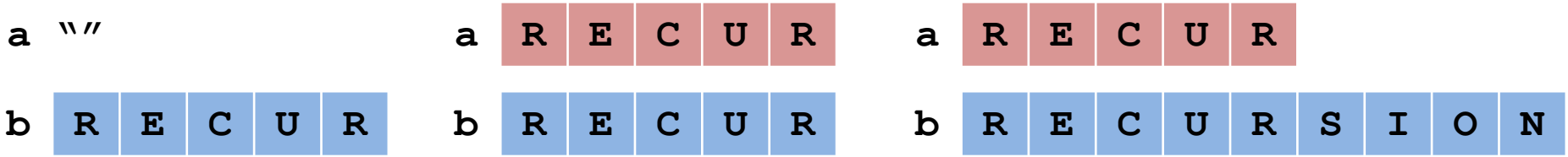
Caso base:

- La **`String a`** es vacía, **`a`** es **subcadena** de **`b`**. **`a`** **`""`**
- La **`String a`** es de longitud mayor que la **`String b`**, **`a`** **no** es **subcadena** de **`b`**.



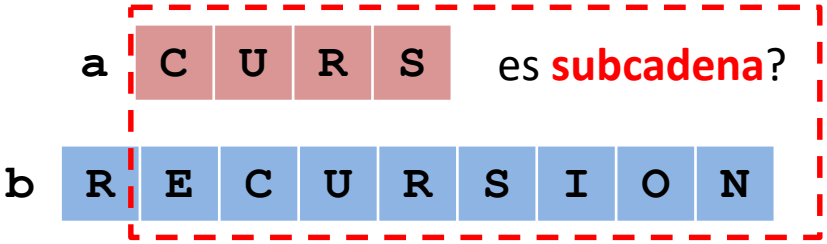
Caso general o recursivo:

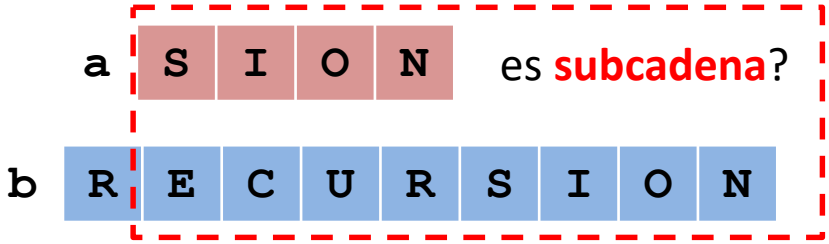
- La **`String a`** es de longitud menor o igual que la **`String b`**, **`a`** es **subcadena** de **`b`**
  - si **`a`** es **prefijo** de **`b`**,



O

  - si **`a`** es **subcadena** de la **substring** de **`b`** desde el carácter **1** en adelante.

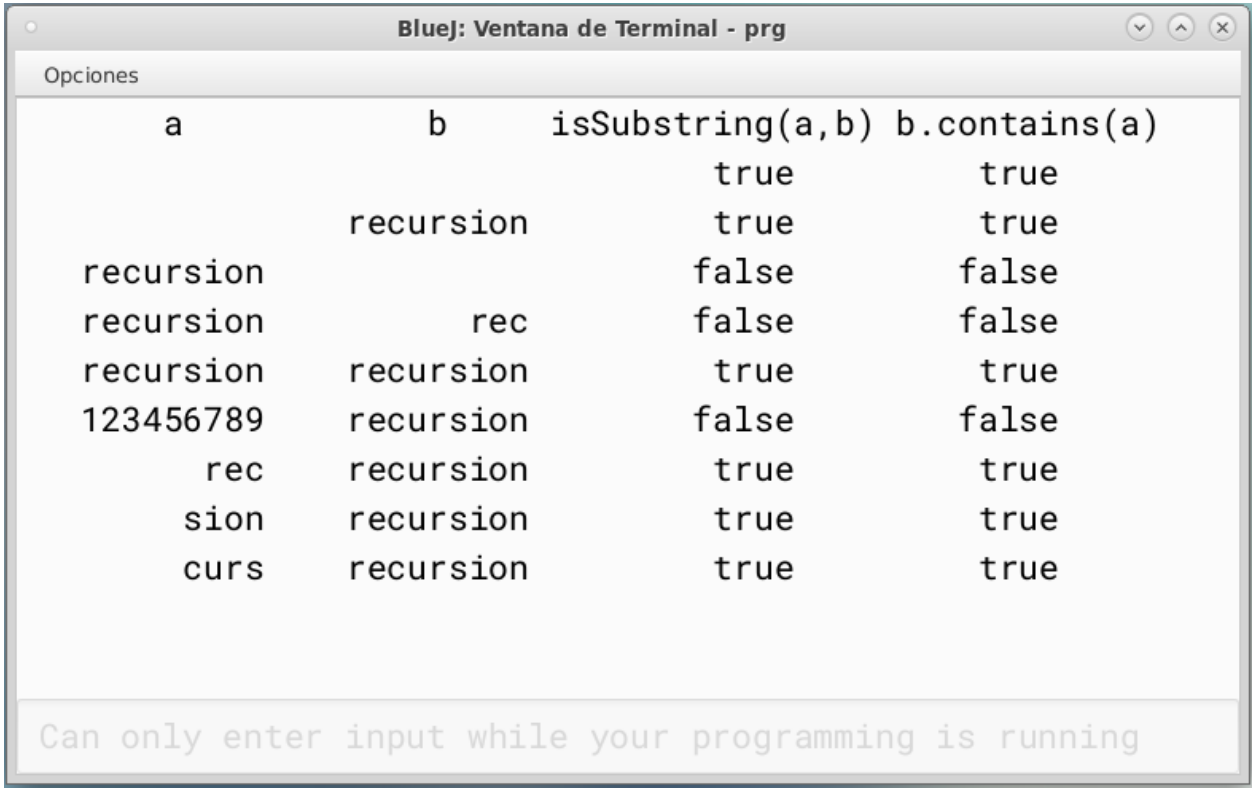




Actividad 5: validación del método isSubstring(String, String)

Escribe una clase programa `TestIsSubstring` que permita ejecutar el método con diferentes datos para comprobar que no hay errores de ejecución y que el resultado que se devuelve en cada caso es el correcto. Como antes, debes identificar las distintas situaciones que se pueden dar, probando que, en todas ellas, el método funciona adecuadamente. En este caso, puedes comparar el resultado con el del método `contains(String)` de la clase `String`.

boolean	**contains**(CharSequence s)	Returns true if and only if this string contains the specified sequence of char values.



```java
public class TestIsSubstring {
    /** No hay objetos de esta clase */
    private TestIsSubstring() { }

    public static void main(String[] args) {
        String[] s = {"", "rec", "pecur", "recurso", "remursi",
                      "123456789", "recursion", "sion", "curs"};

        System.out.printf("%8s %12s %20s %10s\n",
                          "a", "b", "isSubstring(a,b)", "b.contains(a)");
        // a y b vacías
        testIsSubstring(s[0], s[0]);
        // solo a vacía

        // solo b vacía

        // a de mayor longitud que b

        // a y b de igual longitud y a es subcadena de b

        // a y b de igual longitud y a no es subcadena de b

        // a de menor longitud que b y a es sucadena de b
        // porque a es prefijo de b

        // a de menor longitud que b y a es sucadena de b
        // porque a es sufijo de b

        // a de menor longitud que b y a es sucadena de b
        // porque a está en b a partir de una posición intermedia

    }

    private static void testIsSubstring(String a, String b) {

    }
```