Nombre y Apellido: Nº Legajo:

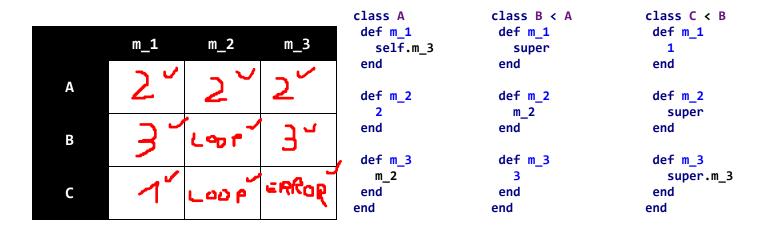
Recuperatorio del Primer Parcial de Programación Orientada a Objetos (72.33) 21/06/2018

Ejercicio 1	Ejercicio 2	Ejercicio 3	Nota	Firma Docente

- ♦ Condición mínima de aprobación: Tener BIEN o BIEN- dos de los tres ejercicios.
- ♦ Las soluciones que no se ajusten al paradigma OO, no serán aceptadas.
- ♦ Las soluciones que no se ajusten estrictamente al enunciado, no serán aceptadas.
- Puede entregarse en lápiz.
- No es necesario escribir las sentencias require.
- ♦ Además de las clases solicitadas se pueden agregar las que consideren necesarias.
- **Escribir en cada hoja Nombre, Apellido, Legajo, Número de Hoja y Total Hojas entregadas.**

Ejercicio 1

Dada la siguiente jerarquía de clases, con los métodos de instancia indicados para cada una, se cuenta con tres instancias homónimas a la clase a la cual pertenecen.



Completar el cuadro de doble entrada (clase y mensaje) indicando qué se obtiene al enviar cada uno de los mensajes a instancias de cada una de las clases.

Ejercicio 2 🗸

Se cuenta con la siguiente familia de clases que representan **elementos** que pueden imprimirse en la consola:

```
module Element
                                                    class TextElement
def contents
                                                     include Element
   raise 'Not Implemented'
                                                     def initialize(text)
end
                                                       @text = text
def to_s
                                                     end
  contents
end
                                                     def contents
                                                       "#{@text}\n"
end
                                                     end
                                                   end
class UniformElement
```

```
class UniformElement
include Element

def initialize(element, width, height)
   @element = element
   @width = width
   @height = height
end
```

```
def contents
   content = ''
   (1..@height).each do
      (1..@width).each do
      content += @element
   end
   content += "\n"
   end
   content
end
end
```

Se quiere dar la posibilidad al usuario de formar **nuevos elementos combinando otros ya existentes**. Para esto se pueden agregar líneas de código a los fuentes de Element, UniformElement y/o TextElement o bien crear clases nuevas, de forma tal que el siguiente programa de prueba imprima lo que se indica a continuación:

```
elem1 = UniformElement.new('+', 6, 2)
                                                   elem6.
elem2 = TextElement.new('hola')
                                                   mundo
elem3 = TextElement.new('mundo')
                                                   +++++
elem4 = UniformElement.new('*', 6, 2)
                                                   +++++
elem5 = TextElement.new('adios')
                                                   hola
elem6 = elem3.above([elem1, elem2])
elem7 = elem3.below([elem4, elem5])
                                                   elem7:
puts "elem6:\n#{elem6}\n"
                                                    *****
puts "elem7:\n#{elem7}\n"
                                                   *****
elem1.element = '.
                                                   adios
elem3.text = 'fin'
                                                   mundo
puts "elem6:\n#{elem6}\n"
puts "elem7:\n#{elem7}\n"
                                                   elem6:
                                                   fin
                                                    . . . . . .
                                                   hola
                                                   elem7:
                                                   *****
                                                   adios
                                                   fin
```

Ejercicio 3

Una aplicación web destinada a empleados de empresas tiene ya definidas las clases Menu y MenuItem. Un MenuItem representa una opción de menú (por ejemplo "Guardar") que cuenta con un menú padre de tipo Menu (por ejemplo "Archivo").

Por defecto, el acceso a los menúes (padres e hijos) es restringido. Para poder acceder a uno de ellos se requiere una autorización. Se desea entonces contar con una forma de otorgar permisos para que los empleados puedan acceder a los ítems para los cuales fueron autorizados.

Un menú puede ser accedido por una o más personas y/o una o más empresas (en este caso, todos los empleados de la empresa pueden acceder al menú). Además un empleado puede pertenecer a una o más empresas.

Para simplificar el mantenimiento, se le puede otorgar el permiso a una persona o empresa directamente a un Menu, y en ese caso podrán acceder a todos los MenuItem correspondientes.

Implementar todo lo necesario para que, con el siguiente programa de prueba,

```
menu1 = Menu.new('File')
menu2 = MenuItem.new('New', menu1)
menu3 = MenuItem.new('Close', menu1)
menu4 = Menu.new('Help')
menu5 = MenuItem.new('About Us', menu4)
menu6 = MenuItem.new('Find...', menu4)

company1 = Company.new('ACME')
company2 = Company.new('Warner')
employee1 = Employee.new('James', [company1])
employee2 = Employee.new('Annie', [company1, company2])
```

```
menu1.authorize(employee1)
menu2.authorize(employee1)
menu5.authorize(employee1)
menu3.authorize(employee2)
menu4.authorize(company2)

[menu1, menu2, menu3, menu4, menu5, menu6].each { |menu| puts menu.access?(employee1) }

puts '#########"

[menu1, menu2, menu3, menu4, menu5, menu6].each { |menu| puts menu.access?(employee2) }
```

se obtenga la siguiente salida:

```
true
true
true
false
true
false
#########
false
false
true
true
true
true
true
true
```