TP N°1: Introducción a POO en Ruby

La siguiente guía cubre los contenidos vistos en la clase teórica:
1. Introducción a POO y Ruby

Para verificar que se cuenta con Ruby instalado ejecutar desde la consola el comando `ruby --version`
Para utilizar la consola de Ruby ejecutar desde la consola el comando `irb`. Para salir de la consola de Ruby invocar a `exit`.

Ejercicio 1

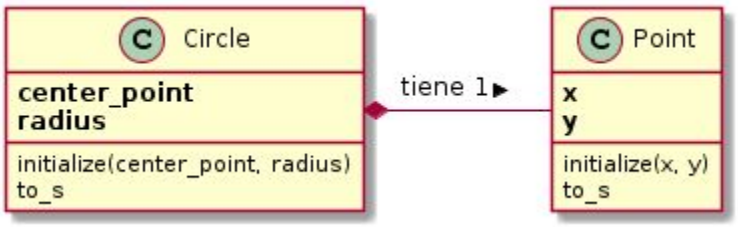
Se cuenta con la clase `Point` que modela un punto en dos dimensiones. A continuación se muestra la implementación de la clase y un programa de prueba. Realizar el diagrama de clases UML correspondiente. ¿Cuál es la utilidad del método `to_s`?

```ruby
# Clase
class Point
 def initialize(x, y)
   @x = x
   @y = y
 end

 def to_s
   "{#{@x},#{@y}}"
 end
end

# Programa de prueba
my_point = Point.new(2,-1)
puts my_point #Imprime {2,-1}
```

darle un formato al texto

Ejercicio 2

Se desea implementar la clase `Circle`. Un círculo se define con un punto en el centro del mismo y un radio. Siguiendo el diagrama de clases que se presenta a continuación, implemente todo lo necesario para que se obtenga la salida esperada del programa de prueba.




```ruby
my_circle = Circle.new(my_point, 3)
puts my_circle #Imprime: Círculo con centro en {2,-1} y radio 3
```

Ejercicio 3

Implementar la clase `ComplexNumber` que modela un número complejo (parte real y parte imaginaria).

La clase además debe permitir que dos instancias de números complejos puedan sumarse retornado una nueva instancia con el resultado de la operación.

Diseñar el UML correspondiente e implementar todo lo necesario para que funcione el siguiente programa de prueba.

```
first_complex = ComplexNumber.new(2, -1)
second_complex = ComplexNumber.new(3, 0)
res_complex = first_complex + second_complex
puts res_complex.real.to_s + ' + ' + res_complex.imaginary.to_s + 'i' # Imprime 5 +
-1i
```

Luego de haber implementado la clase, respecto a la última línea del ejemplo:

a) ¿Qué se obtiene si se quitan las invocaciones a `to_s` de `real` y `imaginary`? Error

b) ¿Dónde corresponde que esté implementado la impresión de un número complejo? En la clase

c) ¿Cómo se puede simplificar la última línea del ejemplo de invocación? Con un to_s

Ejercicio 4

Se desea implementar un método que calcule la distancia entre dos puntos. Analizar cuál de las siguientes opciones es la más apropiada desde el punto de vista del diseño orientado a objetos e implementarla:

a) Agregar un método de instancia `distance(point)` en la clase `Point`.

b) Agregar un método de clase `distance_between(point_a, point_b)` en la clase `Point`.

Ejercicio 5

Se desea implementar la clase `Point3D` para representar puntos en el espacio (x, y, z). Analizar cuál de las siguientes opciones es la más apropiada desde el punto de vista del diseño orientado a objetos e implementarla:

a) Composición: `Point3D` debería contar con una variable de instancia `Point` y una variable de instancia `z`

b) Herencia: `Point3D` debería ser subclase de `Point` y contar con la variable de instancia `z`, (ya que hereda `x` e `y` se heredan de la clase `Point`).

c) Nueva Clase: `Point3D` debería definir tres variables de instancia `x`, `y`, `z`.

Ejercicio 6

Se desea implementar un conjunto de clases que modelan a figuras geométricas en general y a las figuras rectángulo y triángulo en particular. Diseñar las clases `Rectangle` y `Triangle`. Definir en las clases que corresponda los métodos que calculen el perímetro, área, altura y base. ¿Corresponde crear otra clase más? Realizar el diagrama de clases correspondiente.

Nota: Para calcular el área del triángulo, usar la fórmula de Heron.

Ejercicio 7

Se desea agregar un método que determine si una instancia de la clase `Point` pertenece o no al contorno de una figura. ¿Dónde conviene definirlo? Pensar distintas alternativas y quedarse con la mejor desde el punto de vista POO. Hacer que funcione para el rectángulo.

en la figura y que herede

Ejercicio 8

Se desea implementar las clases `Circle` y `Ellipse` como otras dos figuras más de la familia de clases de `Figure`. ¿Debería haber alguna otra relación de herencia entre ambas? Actualizar el diagrama de clases del ***Ejercicio 6***.

Ejercicio 9

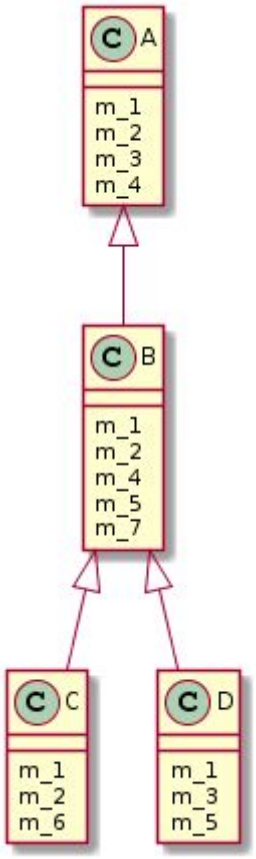
Se desea implementar un conjunto de clases que modelan las cuentas de un banco, sobre las cuales se permiten hacer únicamente depósitos y extracciones. Todas las cuentas tienen un número identificador y un saldo (balance).

Existen dos tipos de cuentas: caja de ahorro y cuenta corriente. La cuenta corriente cuenta con un descubierto y la caja de ahorros no. En otras palabras, un monto puede retirarse de una caja de ahorro sólo si cuenta con los fondos. La cuenta corriente puede tener un saldo negativo.

Diseñar el UML con las clases necesarias, implemente todas las clases y diseñe un programa de prueba que muestre el correcto funcionamiento de las cuentas (instanciar una cuenta, realizar depósitos, extracciones, etc.)

Ejercicio 10

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




```ruby
class A
 def m_1
   m_3
 end

 def m_2
   10
 end

 def m_3
   5
 end

 def m_4
   m_4
 end
end
```

```ruby
class B < A
 def m_1
   8
 end

 def m_2
   super.m_1
 end

 def m_4
   20
 end

 def m_5
   m_3
 end

 def m_7
   super.m_4
 end
end
```

```ruby
class C < B
 def m_1
   super
 end

 def m_2
   m_6
 end

 def m_6
   3
 end
end

class D < B
 def m_1
   super.m_3
 end

 def m_3
   2
 end

 def m_5
   m_4
 end
end
```

Completar el siguiente cuadro indicando qué se obtiene al enviar los mensajes de la primera fila (por separado, no en cascada) a la instancia de la primera columna. Primero hacerlo en papel para verificar que se entendió la invocación a métodos y el uso de la palabra reservada `super`. Luego verificarlo en el entorno Ruby.

	m_1	m_2	m_3	m_4	m_5	m_6	m_7
A	5 ✓	10 ✓	5 ✓	Loop ✓	No Method	No Method	No Method
B	8 ✓	No super-m-o ✓	5 ✓	20 ✓	5 ✓	No Method ✓	No super-m ✓
C	8 ✓	3 ✓	5 ✓	20 ✓	5 ✓	3 ✓	Idem ✓
D	No Method ✓	Nos uper-m-o ✓	2 ✓	29 ✓	20 ✓	No Method ✓	Idem ✓