Clases abstractas

En el POO existe una idea muy sencilla que debería existir las clases abstractas

> Una clase abstracta es una clase que solo sirve para derivar otras clases.

Es decir que una clase abstracta no instancia objetos, no fue diseñada con esa idea, solo está allí para que otras clases hereden de ella. Sin embarjo, el lenguaje Ruby no incorpora tal funcionalidad

> En Ruby no existen las clases abstractas como tal.

La razón de esto tiene que ver con 2 cosas:

1. Ruby soporta algo conocido como "mokey coding".
2. Y que Ruby simplemente no vve la utilidad de tener clases abstractas.

¿Monkey coding?

El *¨"monkey coding"¨* (la "programación estilo chango") es una característica muy propia de Ruby que mermite modificar en tiempo de ejecución el comportamiento de una clase.

Supongamos la siguiente clase:

```
class Punto1D
    def initialize(valX)
        @x=valX
    end
    attr accessor x
    def desplazar(deltaX)
        @x+=deltaX
    end
end
```

Y luego nuestra función principal

```
def ppal
    p1=Punto1D.new(10)
    p1.desplazar(10)
    puts p1.x
end
```

Y luego en el punto de entrada del programa llamamos a la función principal, y luego le modificamos a Punto1D el método desplazar para sumarle deltaX+100 a @x y luego llamamos de nuevo a la función principal:

```
ppal()

# Here we are doing monkey coding
class Punto1D
    def desplazar(deltaX)
        @x+=deltaX+100
    end
end

ppal()
```

Al ejecutar este programa veremos la siguiente salida:

```
20
120
```

El monkey coding es tan solo una forma más de lo que en Ruby se llama *"reapertura de clases"*, que significa que podemos definir una clase, luego si nos da la gana la "volvemos a abrir" para redefinirla o seguir definiendola, y Ruby nos lo permite.

Es decir partiendo de la clase base:

```
class Punto1D
    def initialize(valX)
        @x=valX
    end
    attr accessor x
    def desplazar(deltaX)
        @x+=deltaX
    end
end
```

Podemos utilizar la reapertura de clases para redefinirla en partes:

```

# Opening A
class Punto1D
    def initialize(valX)
        @x=valX
```

```
    end
end

# Opening B
class Punto1D
    attr accessor x
end

# Opening C
class Punto1D
    def desplazar(deltaX)
        @x+=deltaX
    end
end
```

¿Acáso hemos definido tres veces a la clase Punto1D? ¿No hay un problema con esto? Pues Ruby dice que no:

Ruby "une" las tres definiciones de Punto1D en una sola, que funcionará como teníamos la clase original:

```
class Punto1D
    # Opening A
    def initialize(valX)
        @x=valX
    end

    # Opening B
    attr accessor x

    # Opening C
    def desplazar(deltaX)
        @x+=deltaX
    end
end
```

Ruby llama a esto *"reapertura"* porque volver a "abrir" una clase una clase que ya havía sido definida solo para agruegarle algo extra.

Entonces, lo siguiente es importante:

> La reapertura de clases, característica única de Ruby, no puede existir con una clase abstracta.

Y finalmente, Ruby considera que una clase abstracta, así como tal, no es necesaria.

> ¿Quieres una clase "abstracta"? Pues haz una clase y simplemente no hagas instancias de ella.

Y si a alguien esto no le convence, luego sale Ruby con sus *módulos* d emétodos que son algo así como las interfaces de Java pero más flexibles, son un conjunto de métodos vacías que luego una clase que incluya al módulo está obligada a definir.