Ejercicio 2 Template Method

a) Template Method en Magnitude, categoría comparing

La clase Magnitude es abstracta, tiene como subclase a Number, que a su vez tiene como subclases a Float, Integer, Fraction, etc.
En la clase Magnitude, tenemos un método llamado #min:max:

```
min: maxValue max: minValue
        "Take the minimum between self and maxValue, then the maximum with
minValue"
        "(10 min: 20 max: 5) >>> 10"
        "(10 min: 20 max: 11) >>> 11"
        "(24 min: 20 max: 5) >>> 20"

        ^ (self min: maxValue) max: minValue
```

Este metodo se envia a una magnitud, con otras dos magnitudes como parametros
Devuelve siempre la magnitud que se encuentra entre medio de las otras dos
Ahora bien dentro de ese método, hace un llamado a otros métodos #min: y #max:

```
min: aMagnitude
        "Answer the receiver or the
argument, whichever has the lesser
        magnitude."

        self < aMagnitude
                ifTrue: [^self]
                ifFalse: [^aMagnitude]
```

```
max: aMagnitude
        "Answer the receiver or the
argument, whichever has the greater
        magnitude."

        self > aMagnitude
                ifTrue: [^self]
                ifFalse: [^aMagnitude]
```

Estos, a su vez usan los mensajes #> y #< para comparar si una magnitud es mayor o menor que la otra (segun corresponda)

```
> aMagnitude
        "Answer whether the receiver is
greater than the argument."

        ^aMagnitude < self
```

```
< aMagnitude
        "Answer whether the receiver is less
than the argument."

        ^self subclassResponsibility
```

Como vemos, el mensaje #> simplemente llama al mensaje #< pero invirtiendo el objeto receptor por el objeto enviado como parametro, es decir:

- ^objetoUno>objetoDos es lo mismo que ^objetoDos<objetoUno

La implementación del mensaje #< es delegada a las subclases (mencionadas mas arriba).

Esto es porque los criterios para comparar magnitudes varian segun el tipo de magnitud que sea (enteros, flotantes, fracciones, etc).
Pero el esqueleto es siempre el mismo, y ese comportamiento en común es lo que vemos en los mensajes #min:max:, #min:, #max: y #>.
Luego lo que cambia dependiendo de la sublase es la implementación del mensaje #<, como vemos en el ejemplo de abajo:

Clase Integer

```
< aNumber
        aNumber isInteger ifTrue:
                [self negative == aNumber negative
                        ifTrue: [self negative
                                        ifTrue: [^ (self bytesCompare: aNumber) > 0]
                                        ifFalse: [^ (self bytesCompare: aNumber) < 0]]
                        ifFalse: [^ self negative]].
        ^ aNumber adaptToInteger: self andCompare: #<
```

Clase Fraction

```
< aNumber
        aNumber isFraction ifTrue:
                [^ numerator * aNumber denominator < (aNumber numerator *
denominator)].
        ^ aNumber adaptToFraction: self andCompare: #<
```

Lo mismo para el resto de las subclases.

El profesor:

- Señaló que el mensaje = hace algo parecido.
- Magnitude es interesante porque tiene muchos template methods implementados. Simplemente implementando el < y el =, hereda un montón de métodos “gratis”, ya que esos métodos están basados en el < e =. Por ejemplo el between:and:, min:max, etc.

en el otro ver el do: y el select:
el do: depende de cada una de las subclases.

b) Template Method en Collection, categoría accessing

Collection es abstracta y tiene como subclases a otras clases abstractas como HashedCollection, SequenceableCollection y ArrayedCollection, luego estas tienen más subclases.
En la categoría “accessing” tenemos 3 métodos: #anyOne #capacity #size.

#anyOne devuelve cualquier elemento de la colección, simplemente agarra el primer elemento que se cruza. Antes de esto, realiza una validación para asegurarse que la colección tiene al menos 1 elemento, esto lo hace con el método #emptyCheck. Este a su vez, usa el mensaje **#size**.

También notamos que #capacity usa el mensaje **#size** para ser calculado.
En este mensaje notamos que se hace uso de Template Method: #size tiene un código por defecto en la clase Collection, pero es sobreescrito por sus subclases, ya que cada una tiene formas diferentes o más eficientes de calcular #size.