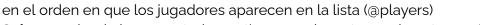
**Ejercicio 1.** El <u>Dr Nefario</u> está creando un nuevo juego de cartas para jugar con los minions. Todavía no lo tiene definido al 100%, pero de algo está seguro, que durante una partida (Game) se repartirán las cartas (Card) de un mismo mazo (Deck) a N jugadores de diferente forma a lo largo del juego. La estrategia de repartición de cartas podrá cambiar a lo largo de una misma partida. Existen tres formas de repartir las cartas:

- En bloque, donde la computadora retira 5 cartas del mazo y se las entrega al primer jugador. Posteriormente, retira otras 5 cartas del mazo y se las entrega al segundo jugador, y así sucesivamente hasta que todos los jugadores hayan recibido 5 cartas. Se reparten las cartas en el orden en que los jugadores aparecen en la lista (@players)
- Uno por uno, la computadora retira una carta y la entrega al primer jugador. Posteriormente, retira otra carta y se la entrega al segundo jugador y así sucesivamente hasta que todos hayan recibido una carta. Se reparten las cartas





• **Solo uno**, donde la computadora retira una sola carta y se la entrega aleatoriamente a un jugador. Los demás jugadores no reciben ninguna carta.

El <u>Dr. Nefario</u> realizó un pequeño modelo orientado a objeto y el método <u>split</u> que permite repartir las cartas **en bloque** a N jugadores. El <u>Dr. Nefario</u> se enteró que en el curso de Ing. de Software se estudian patrones de diseño, por lo que decidió delegar la tarea de implementar las otras formas de repartir cartas a los estudiantes del curso. Su única condición es que modifiquen su clase *Game* y utilicen el patrón *strategy* de forma tal que el pueda agregar nuevas formas de repartir cartas en el futuro de forma fácil, iterativa e incremental (**extensible!**).

- A. Implemente el patrón *strategy*. Modifique la clase Game, y agregue las clases necesarias para implementar el patrón *strategy*. Su solución debe implementar las tres estrategias de repartición escritas: en bloque, uno por uno, solo uno.
- B. Escriba un pequeño código de ejemplo donde se repartan cartas a 3 jugadores, de la siguiente forma: (i) primero se reparten 5 cartas a cada jugador (**en bloque**), (ii), se reparte 1 carta a cada jugador (**uno por uno**); (iii) se repare una carta a un jugador al azar (**sólo uno**). Note que todas las reparticiones se deben hacer con el mismo mazo (*Deck*) en la misma partida (Game).

Código del Dr. Nefario

```
class Card
                                class Player
 attr reader :value
                                   def initialize(id)
  def initialize (value,
                                       @id = id
kind)
                                       @cards =[]
    @value = value
                                   end
    @kind = kind
                                   def add(card)
  end
                                       @cards.push(card)
 def to s
                                   end
   return
                                   def printCards()
"#{@value}#{@kind}"
                                       puts "Player #{@id}: #{@cards.join(",")}"
                                   end
                                end
end
class CardDeck
  def initialize
       @cards = []
       ["H", "T", "D", "S"].each do
           |kind|
           (1..13).each do |number|
                 @cards.push(Card.new(number,kind))
           end
       end
   end
   def shuffle
       @cards.shuffle
   end
  def pickRandom
       randomCard = @cards.sample
       @cards.delete(randomCard)
       return randomCard
   end
   def pickLast
       return @cards.pop
   end
end
class Game
   def initialize(numberOfPlayers)
       @deck = CardDeck.new
       @players = []
       numberOfPlayers.times { |id| @players.push(Player.new(id))}
   end
   def printState()
       @players.each do |player|
           player.printCards
       end
   end
   def split()
       # reparte 5 cartas a cada jugador
       @players.each do |player|
          5.times{
               player.add(@deck.pickLast)
       end
   end
end
```

## Ejercicio 2. Considere la clase Batería

```
class Bateria
  def initialize
    @carga = 100 # 100 % de carga
    @tiempo = 60 # 60 minutos restantes
  end
  def consume(voltios)
    porcentaje_consumido = voltios/256
    @carga = @carga - porcentaje_consumido
    @tiempo = @tiempo - porcentaje_consumido*60
  end
end
```

Aplique el patrón observer, creando dos observadores: (1) un observador que muestre una alerta cuando la batería tiene menos del 20% de carga; (2) un observador que cuando la batería tenga o de carga imprima el mensaje "bye bye". El diseño de clases debe permitir agregar fácilmente nuevos tipos de observadores (extensible).