Solución Ejercicio 1.

```ruby
class Splitter
   def split(deck, players)
       raise NotImplementedError
   end
end
```

```ruby
class BlockSplitter
   def split(deck, players)
       players.each do  |player|
           5.times{
           player.add(deck.pickLast)
           }
       end
   end
end
```

```ruby
class OneAllSplitter
   def split(deck, players)
       players.each do  |player|
           player.add(deck.pickLast)
       end
   end
end
```

```ruby
class OnlyOneRandomSplitter
   def split(deck, players)
       players.sample.add(deck.pickRandom)
   end
end
```

```ruby
class Game
   def split(splitter)
       splitter.split(@deck,@players)
   end
end
```

```ruby
game = Game.new(3)
game.shuffle()
game.split(BlockSplitter.new)
game.split(OneAllSplitter.new)
game.split(OnlyOneRandomSplitter.new)
```

Puede variar la implementación un poco, pero la idea es que las estrategias cambian en tiempo de ejecución, y sean aplicadas al mismo objeto game. De esta forma se reparte utilizando el mismo mazo.

Solución Ejercicio 2.

```ruby
class Observer
   # la clase bateria no debe saber quien la observa
   # el metodo update recibe la bateria
   # porque no sabe que información necesitaran los objetos que lo observaran
   def update(battery)
      raise NotImplementedError
   end
end

class LowBatteryObserver < Observer
   def update(battery)
      # un problema es que la clase bateria tiene que tener accesores para que los
observadores consulten su información
      if battery.carga() < 20
         puts 'low battery'
      end
   end
```

```
end
class PowerOffObserver < Observer
    def update(battery)
        if battery.carga() == 0
            puts 'bye bye'
        end
    end
end
```

```
class Battery
  def initialize
    super
    @carga = 100 # 100 % de carga
    @tiempo = 60 # 60 minutos restantes
    @observers = []
  end
  def add(obs)
    @observers.push(obs)
  end
  #se llama al metodo notify all normalmente cuando el objeto cambia de estado
  def notifyAll()
    @observers.each do |obs|
      # manda una referencia a si misma para que cada observador acceda a la
información que necesite
      obs.update(self)
    end
  end
  def consume(voltios)
    porcentaje_consumido = voltios/256
    @carga = @carga - porcentaje_consumido
    @tiempo = @tiempo - porcentaje_consumido*60
    # la clase observada notifica su cambio de estado a los observadores
    notifyAll()
  end
  def carga
    return @carga
  end
end
```

```
b = Battery.new
b.register(LowBatteryObserver.new)
b.register(NoBatteryObserver.new)

b.consume(10)
b.consume(10)
b.consume(10)
```