Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

# Clase 11: Clean Code (Cap. 8)

**Rodrigo Toro Icarte** (rodrigo.toro@ing.puc.cl)

IIC2113 Diseño Detallado de Software

**14 de Septiembre, 2023**

# En la Clase de Hoy

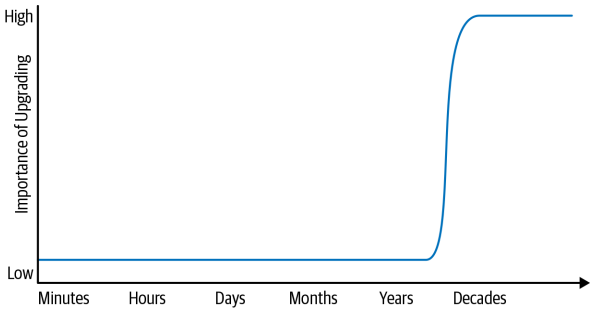
# Boundaries

by James Grenning

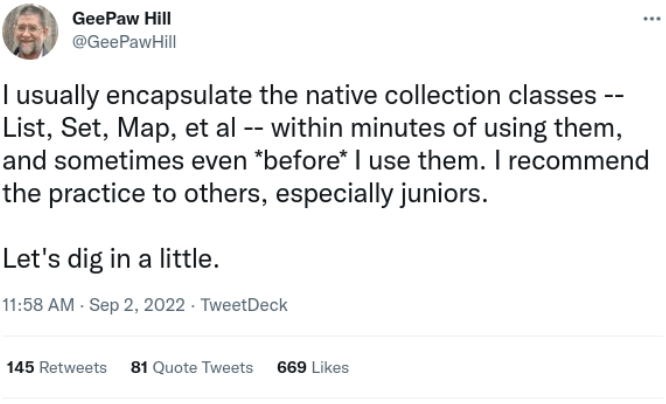
# Boundaries

Piensa en el código que hayas programado con mayor expectativa de vida.

- ¿Qué dependencias tiene?
- ¿Qué librerías usaste?
- ¿Encapsulaste bien su comportamiento cosa que sea fácil actualizarlas?



# Boundaries



**GeePaw Hill**
@GeePawHill

I usually encapsulate the native collection classes -- List, Set, Map, et al -- within minutes of using them, and sometimes even *before* I use them. I recommend the practice to others, especially juniors.

Let's dig in a little.

11:58 AM · Sep 2, 2022 · TweetDeck

**145** Retweets **81** Quote Tweets **669** Likes

`https://twitter.com/GeePawHill/status/1565730818424717312`

# Boundaries

Hoy en día nadie programa desde cero. Todos usamos librerías escritas por alguien más. Esas librerías tienen *interfaces* bien definidas que nos permiten usarlas.

**¿Cuál es el problema?**

# Boundaries

Hoy en día nadie programa desde cero. Todos usamos librerías escritas por alguien más. Esas librerías tienen *interfaces* bien definidas que nos permiten usarlas.

**¿Cuál es el problema?**

1. Normalmente la interfaz del proveedor no se ajusta del todo a los que necesitamos.
2. Cuando el proveedor cambie su interfaz destruirá nuestro código.

# Boundaries

Hoy en día nadie programa desde cero. Todos usamos librerías escritas por alguien más. Esas librerías tienen *interfaces* bien definidas que nos permiten usarlas.

**¿Cuál es el problema?**

1. Normalmente la interfaz del proveedor no se ajusta del todo a los que necesitamos.
2. Cuando el proveedor cambie su interfaz destruirá nuestro código.

**Ejemplo**: Necesitas modelar una lista de cartas y decides usar un `ArrayList`.

```
2 ArrayList cartas = new ArrayList();
```

# Boundaries

Hoy en día nadie programa desde cero. Todos usamos librerías escritas por alguien más. Esas librerías tienen *interfaces* bien definidas que nos permiten usarlas.

**¿Cuál es el problema?**

1. Normalmente la interfaz del proveedor no se ajusta del todo a los que necesitamos.
2. Cuando el proveedor cambie su interfaz destruirá nuestro código.

**Ejemplo**: Necesitas modelar una lista de cartas y decides usar un `ArrayList`.

```
2 ArrayList cartas = new ArrayList();
```

Este `ArrayList` es clave en tu programa por lo que se encuentra en **todos** lados.

# Boundaries

Hoy en día nadie programa desde cero. Todos usamos librerías escritas por alguien más. Esas librerías tienen *interfaces* bien definidas que nos permiten usarlas.

**¿Cuál es el problema?**

1. Normalmente la interfaz del proveedor no se ajusta del todo a los que necesitamos.
2. Cuando el proveedor cambie su interfaz destruirá nuestro código.

**Ejemplo**: Necesitas modelar una lista de cartas y decides usar un `ArrayList`.

```
2 ArrayList cartas = new ArrayList();
```

Este `ArrayList` es clave en tu programa por lo que se encuentra en **todos** lados.

¿Qué podría salir mal?

# Boundaries

**1)** Quienes programaron `ArrayList` querían que su código le sirviera a todo el mundo:

# Boundaries

**1)** Quienes programaron `ArrayList` querían que su código le sirviera a todo el mundo:

- Adapter(IList)
- Add(Object)
- AddRange(ICollection)
- BinarySearch(Int32, Int32, Object, IComparer)
- BinarySearch(Object)
- BinarySearch(Object, IComparer)
- Clear()
- Clone()
- Contains(Object)
- CopyTo(Array)
- CopyTo(Array, Int32)
- CopyTo(Int32, Array, Int32, Int32)
- Equals(Object)
- FixedSize(ArrayList)
- FixedSize(IList)
- GetEnumerator()
- GetEnumerator(Int32, Int32)
- GetHashCode()
- GetRange(Int32, Int32)
- GetType()
- IndexOf(Object)
- IndexOf(Object, Int32)
- IndexOf(Object, Int32, Int32)
- Insert(Int32, Object)
- InsertRange(Int32, ICollection)
- LastIndexOf(Object)
- LastIndexOf(Object, Int32)
- LastIndexOf(Object, Int32, Int32)
- MemberwiseClone()
- ReadOnly(ArrayList)
- ReadOnly(IList)
- Remove(Object)
- RemoveAt(Int32)
- RemoveRange(Int32, Int32)
- Repeat(Object, Int32)
- Reverse()
- Reverse(Int32, Int32)
- SetRange(Int32, ICollection)
- Sort()
- Sort(IComparer)
- Sort(Int32, Int32, IComparer)
- Synchronized(ArrayList)
- Synchronized(IList)
- ToArray()
- ToArray(Type)
- ToString()
- TrimToSize()

# Boundaries

**1)** Quienes programaron `ArrayList` querían que su código le sirviera a todo el mundo:

- Adapter(IList)
- Add(Object)
- AddRange(ICollection)
- BinarySearch(Int32, Int32, Object, IComparer)
- BinarySearch(Object)
- BinarySearch(Object, IComparer)
- Clear()
- Clone()
- Contains(Object)
- CopyTo(Array)
- CopyTo(Array, Int32)
- CopyTo(Int32, Array, Int32, Int32)
- Equals(Object)
- FixedSize(ArrayList)
- FixedSize(IList)
- GetEnumerator()
- GetEnumerator(Int32, Int32)
- GetHashCode()
- GetRange(Int32, Int32)
- GetType()
- IndexOf(Object)
- IndexOf(Object, Int32)
- IndexOf(Object, Int32, Int32)
- Insert(Int32, Object)
- InsertRange(Int32, ICollection)
- LastIndexOf(Object)
- LastIndexOf(Object, Int32)
- LastIndexOf(Object, Int32, Int32)
- MemberwiseClone()
- ReadOnly(ArrayList)
- ReadOnly(IList)
- Remove(Object)
- RemoveAt(Int32)
- RemoveRange(Int32, Int32)
- Repeat(Object, Int32)
- Reverse()
- Reverse(Int32, Int32)
- SetRange(Int32, ICollection)
- Sort()
- Sort(IComparer)
- Sort(Int32, Int32, IComparer)
- Synchronized(ArrayList)
- Synchronized(IList)
- ToArray()
- ToArray(Type)
- ToString()
- TrimToSize()

Esto es demasiada libertad. No quiero dejar que mi compañero haga `cartas.clear()` para sacar su feature adelante matando mi código 😅

# Boundaries

**2)** Al mismo tiempo, `ArrayList` no hace exactamente lo que queremos.

# Boundaries

**2)** Al mismo tiempo, `ArrayList` no hace exactamente lo que queremos.

```
3 Carta carta = (Carta) cartas[idCarta];
```

... esta conversión se encuentra en **todo** el código.

# Boundaries

**2)** Al mismo tiempo, `ArrayList` no hace exactamente lo que queremos.

```
3 Carta carta = (Carta) cartas[idCarta];
```

... esta conversión se encuentra en **todo** el código.

**3)** Si la interfaz de `ArrayList` cambia o si descubrimos que usar `List` era mejor que usar `ArrayList` tendremos que cambiar esto en todo nuestro código.

# Boundaries

**2)** Al mismo tiempo, `ArrayList` no hace exactamente lo que queremos.

```
3 Carta carta = (Carta) cartas[idCarta];
```

... esta conversión se encuentra en **todo** el código.

**3)** Si la interfaz de `ArrayList` cambia o si descubrimos que usar `List` era mejor que usar `ArrayList` tendremos que cambiar esto en todo nuestro código.

# Boundaries

**2)** Al mismo tiempo, `ArrayList` no hace exactamente lo que queremos.

```
3 Carta carta = (Carta) cartas[idCarta];
```

... esta conversión se encuentra en **todo** el código.

**3)** Si la interfaz de `ArrayList` cambia o si descubrimos que usar `List` era mejor que usar `ArrayList` tendremos que cambiar esto en todo nuestro código.

# Boundaries

**2)** Al mismo tiempo, `ArrayList` no hace exactamente lo que queremos.

```
3 Carta carta = (Carta) cartas[idCarta];
```

... esta conversión se encuentra en **todo** el código.

**3)** Si la interfaz de `ArrayList` cambia o si descubrimos que usar `List` era mejor que usar `ArrayList` tendremos que cambiar esto en todo nuestro código.

# Boundaries

**Solución**: Crear nuestra propia interfaz *a la medida* de lo que necesitamos.

```
2  public class Cartas {
3      private ArrayList cartas = new ArrayList();
4
5      public void AgregarCarta(Carta carta)
6          => cartas.Add(carta);
7
8      public Carta ObtenerCarta(int id)
9          => (Carta)cartas[id];
10     // ...
11 }
```

# Boundaries

**Solución**: Crear nuestra propia interfaz *a la medida* de lo que necesitamos.

```
2  public class Cartas {
3      private ArrayList cartas = new ArrayList();
4
5      public void AgregarCarta(Carta carta)
6          => cartas.Add(carta);
7
8      public Carta ObtenerCarta(int id)
9          => (Carta)cartas[id];
10     // ...
11 }
```

**Ventajas**:

- Si cambia la interfaz de `ArrayList` (o queremos cambiar `ArrayList` por otra cosa) solo tenemos que cambiar la clase `Cartas`.
- Solo proveemos los métodos que nos sirven (y ocultamos el resto).

# Boundaries

*"We are not suggesting that every use of* `Map` *be encapsulated in this form. Rather, we are advising you not to pass* `Maps` *(or any other interface at a boundary) around your system. If you use a boundary interface like* `Map`*, keep it inside the class, or close family of classes, where it is used. Avoid returning it from, or accepting it as an argument to, public APIs."*

# Exploring and Learning Boundaries

El capítulo incluye una idea más que vale la pena discutir.

# Exploring and Learning Boundaries

El capítulo incluye una idea más que vale la pena discutir. Digamos que quieres usar una librería y estás aprendiendo a usarla. Por ejemplo, `ImageSharp`.

# Exploring and Learning Boundaries

El capítulo incluye una idea más que vale la pena discutir. Digamos que quieres usar una librería y estás aprendiendo a usarla. Por ejemplo, `ImageSharp`.

Leemos la documentación y comenzamos a experimentar con la librería.

```
2 string imagePath = "Tests/Lenna.png";
3 string savePath = "Tests/Resultado.png";
4 Image<Rgb24> image = Image.Load<Rgb24>(imagePath);
5 image.Save(savePath);
```

# Exploring and Learning Boundaries

El capítulo incluye una idea más que vale la pena discutir. Digamos que quieres usar una librería y estás aprendiendo a usarla. Por ejemplo, `ImageSharp`.

Leemos la documentación y comenzamos a experimentar con la librería.

```
string imagePath = "Tests/Lenna.png";
string savePath = "Tests/Resultado.png";
Image<Rgb24> image = Image.Load<Rgb24>(imagePath);
image.Save(savePath);
```

Revisamos la carpeta `Tests` y efectivamente `Resultado.png` está ahí. ¡Vamos bien!

# Exploring and Learning Boundaries

Ahora intentamos editar un poco la imagen.

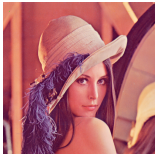
```
string imagePath = "Tests/Lenna.png";
string savePath = "Tests/Resultado.png";
Image<Rgb24> image = Image.Load<Rgb24>(imagePath);
image[20, 30] = new Rgb24(0, 255, 0);
image[21, 30] = new Rgb24(0, 255, 0);
image[20, 31] = new Rgb24(0, 255, 0);
image[21, 31] = new Rgb24(0, 255, 0);
image.Save(savePath);
```

# Exploring and Learning Boundaries

Ahora intentamos editar un poco la imagen.

```
2 string imagePath = "Tests/Lenna.png";
3 string savePath = "Tests/Resultado.png";
4 Image<Rgb24> image = Image.Load<Rgb24>(imagePath);
5 image[20, 30] = new Rgb24(0, 255, 0);
6 image[21, 30] = new Rgb24(0, 255, 0);
7 image[20, 31] = new Rgb24(0, 255, 0);
8 image[21, 31] = new Rgb24(0, 255, 0);
9 image.Save(savePath);
```

**Vemos el resultado**:

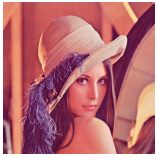
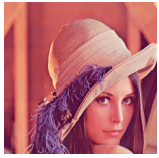


# Exploring and Learning Boundaries

Ahora intentamos editar un poco la imagen.

```
string imagePath = "Tests/Lenna.png";
string savePath = "Tests/Resultado.png";
Image<Rgb24> image = Image.Load<Rgb24>(imagePath);
image[20, 30] = new Rgb24(0, 255, 0);
image[21, 30] = new Rgb24(0, 255, 0);
image[20, 31] = new Rgb24(0, 255, 0);
image[21, 31] = new Rgb24(0, 255, 0);
image.Save(savePath);
```

**Vemos el resultado**:

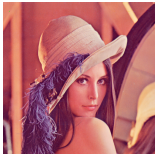
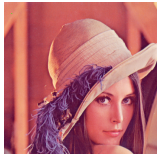
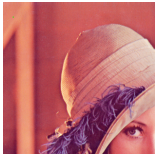



# Exploring and Learning Boundaries

Ahora intentamos editar un poco la imagen.

```
2 string imagePath = "Tests/Lenna.png";
3 string savePath = "Tests/Resultado.png";
4 Image<Rgb24> image = Image.Load<Rgb24>(imagePath);
5 image[20, 30] = new Rgb24(0, 255, 0);
6 image[21, 30] = new Rgb24(0, 255, 0);
7 image[20, 31] = new Rgb24(0, 255, 0);
8 image[21, 31] = new Rgb24(0, 255, 0);
9 image.Save(savePath);
```

**Vemos el resultado**:

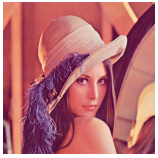
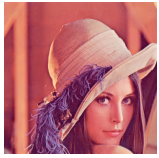
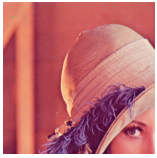
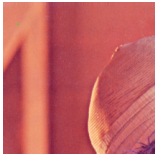




# Exploring and Learning Boundaries

Ahora intentamos editar un poco la imagen.

```
string imagePath = "Tests/Lenna.png";
string savePath = "Tests/Resultado.png";
Image<Rgb24> image = Image.Load<Rgb24>(imagePath);
image[20, 30] = new Rgb24(0, 255, 0);
image[21, 30] = new Rgb24(0, 255, 0);
image[20, 31] = new Rgb24(0, 255, 0);
image[21, 31] = new Rgb24(0, 255, 0);
image.Save(savePath);
```

**Vemos el resultado**:

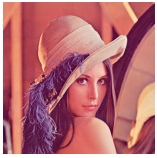
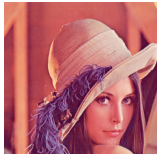
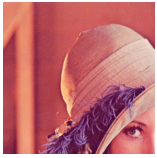
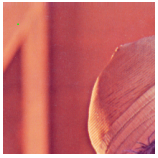





# Exploring and Learning Boundaries

Ahora intentamos editar un poco la imagen.

```
2 string imagePath = "Tests/Lenna.png";
3 string savePath = "Tests/Resultado.png";
4 Image<Rgb24> image = Image.Load<Rgb24>(imagePath);
5 image[20, 30] = new Rgb24(0, 255, 0);
6 image[21, 30] = new Rgb24(0, 255, 0);
7 image[20, 31] = new Rgb24(0, 255, 0);
8 image[21, 31] = new Rgb24(0, 255, 0);
9 image.Save(savePath);
```

**Vemos el resultado**:

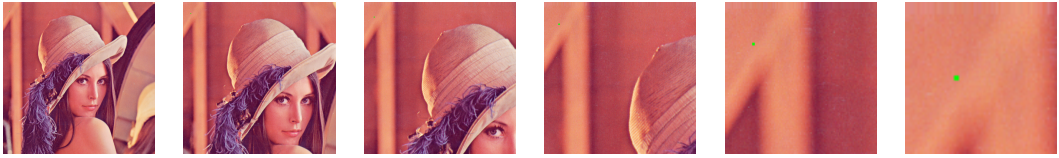






# Exploring and Learning Boundaries

Ahora intentamos editar un poco la imagen.

```
string imagePath = "Tests/Lenna.png";
string savePath = "Tests/Resultado.png";
Image<Rgb24> image = Image.Load<Rgb24>(imagePath);
image[20, 30] = new Rgb24(0, 255, 0);
image[21, 30] = new Rgb24(0, 255, 0);
image[20, 31] = new Rgb24(0, 255, 0);
image[21, 31] = new Rgb24(0, 255, 0);
image.Save(savePath);
```

**Vemos el resultado**:



¡Ahí está nuestro punto verde! :D

# Exploring and Learning Boundaries

Ya, pero aquí viene la observación brillante:

# Exploring and Learning Boundaries

Ya, pero aquí viene la observación brillante:

```
string imagePath = "Tests/Lenna.png";
string savePath = "Tests/Resultado.png";
Image<Rgb24> image = Image.Load<Rgb24>(imagePath);
image[20, 30] = new Rgb24(0, 255, 0);
image[21, 30] = new Rgb24(0, 255, 0);
image[20, 31] = new Rgb24(0, 255, 0);
image[21, 31] = new Rgb24(0, 255, 0);
image.Save(savePath);
```

# Exploring and Learning Boundaries

Ya, pero aquí viene la observación brillante:

```
string imagePath = "Tests/Lenna.png";
string savePath = "Tests/Resultado.png";
Image<Rgb24> image = Image.Load<Rgb24>(imagePath);
image[20, 30] = new Rgb24(0, 255, 0);
image[21, 30] = new Rgb24(0, 255, 0);
image[20, 31] = new Rgb24(0, 255, 0);
image[21, 31] = new Rgb24(0, 255, 0);
image.Save(savePath);
```

Esto es prácticamente un test que podríamos llegar y poner en nuestra batería de tests.

# Exploring and Learning Boundaries

```
2  public class ImageSharpTests {
3      [Fact]
4      public void LeerGuardarEditarUnaImagen() {
5          string imagePath = "Tests/Lenna.png";
6          string savePath = "Tests/Resultado.png";
7          Image<Rgb24> image = Image.Load<Rgb24>(imagePath);
8          image[20, 30] = new Rgb24(0, 255, 0);
9          image.Save(savePath);
10
11         Image<Rgb24> savedImage = Image.Load<Rgb24>(savePath);
12         Assert.Equal(0, savedImage[20, 30].R);
13         Assert.Equal(255, savedImage[20, 30].G);
14         Assert.Equal(0, savedImage[20, 30].B);
15     }
16 }
```

# Exploring and Learning Boundaries

```
public class ImageSharpTests {
    [Fact]
    public void LeerGuardarEditarUnaImagen() {
        string imagePath = "Tests/Lenna.png";
        string savePath = "Tests/Resultado.png";
        Image<Rgb24> image = Image.Load<Rgb24>(imagePath);
        image[20, 30] = new Rgb24(0, 255, 0);
        image.Save(savePath);

        Image<Rgb24> savedImage = Image.Load<Rgb24>(savePath);
        Assert.Equal(0, savedImage[20, 30].R);
        Assert.Equal(255, savedImage[20, 30].G);
        Assert.Equal(0, savedImage[20, 30].B);
    }
}
```

Esto no solo es un test *gratis*, también es un test que nos avisará cuando alguna actualización de la librería pueda romper nuestro código ✨

# Discusión

# Discusión

*"Interesting things happen at boundaries. Change is one of those things. Good software designs accommodate change without huge investments and rework. When we use code that is out of our control, special care must be taken to protect our investment and make sure future change is not too costly."*

# Discusión

*"Interesting things happen at boundaries. Change is one of those things. Good software designs accommodate change without huge investments and rework. When we use code that is out of our control, special care must be taken to protect our investment and make sure future change is not too costly."*

*"Code at the boundaries needs clear separation and tests that define expectations. We should avoid letting too much of our code know about the third-party particulars. It's better to depend on something you control than on something you don't control, lest it end up controlling you."*

# ¿Cómo afecta esto a la E3?

# ¿Cómo afecta esto a la E3?

**Preguntas clave**:

# ¿Cómo afecta esto a la E3?

**Preguntas clave**:

- ¿Qué dependencias externas tiene tu proyecto?

# Analizando Entregas del Proyecto

# Analizando Entregas del Proyecto

```
1  public class Player
2  {
3    public Deck deck;
4    public List<Card> handCards = new List<Card>();
5    public List<Card> arsenalCards = new List<Card>();
6    public List<Card> ringsideCards = new List<Card>();
7    public List<Card> ringAreaCards = new List<Card>();
8
9    public SuperStar superstar;
10   public int fortitudeRating = 0;
11
12   /* ... */
13 }
```

# Analizando Entregas del Proyecto

```
1  public class Player
2  {
3    public Deck deck;
4    public List<Card> handCards = new List<Card>();
5    public List<Card> arsenalCards = new List<Card>();
6    public List<Card> ringsideCards = new List<Card>();
7    public List<Card> ringAreaCards = new List<Card>();
8
9    public SuperStar superstar;
10   public int fortitudeRating = 0;
11
12   /* ... */
13 }
```

¿Le descontamos por tener los atributos públicos?

# Analizando Entregas del Proyecto

Depende de si `Player` es un objeto o una estructura de datos.

```
1  public class Player
2  {
3    public Deck deck;
4    public List<Card> handCards = new List<Card>();
5    public List<Card> arsenalCards = new List<Card>();
6    public List<Card> ringsideCards = new List<Card>();
7    public List<Card> ringAreaCards = new List<Card>();
8
9    public SuperStar superstar;
10   public int fortitudeRating = 0;
11
12   public void PutDeckCardsOnTable() {/*...*/ }
13   public void DrawCardsfromArsenal(int n) { /*...*/ }
14 }
```

# Analizando Entregas del Proyecto

Depende de si `Player` es un objeto o una estructura de datos.

```
1  public class Player
2  {
3    public Deck deck;
4    public List<Card> handCards = new List<Card>();
5    public List<Card> arsenalCards = new List<Card>();
6    public List<Card> ringsideCards = new List<Card>();
7    public List<Card> ringAreaCards = new List<Card>();
8
9    public SuperStar superstar;
10   public int fortitudeRating = 0;
11
12   public void PutDeckCardsOnTable() {/*...*/ }
13   public void DrawCardsfromArsenal(int n) { /*...*/ }
14 }
```

... tiene pinta de clase.

# Analizando Entregas del Proyecto

Veamos este código:

```
public class Player {
  public List<Card> handCards = new List<Card>();
  public List<Card> arsenalCards = new List<Card>();
  public List<Card> ringsideCards = new List<Card>();
  public List<Card> ringAreaCards = new List<Card>();
  /* ... */
  public void DrawCardsfromArsenal(int n) {
    for (int i = 0; i < n; i++) {
      //Obtiene la última carta
      Card cardToTransfer = arsenalCards.Last();
      //Añade la carta a handCards
      handCards.Add(cardToTransfer);
      //Elimina la última carta de arsenalCards
      arsenalCards.RemoveAt(arsenalCards.Count - 1);
    }
  }
}
```

# Analizando Entregas del Proyecto

**Cap 8**: `List` debería ser encapsulado.

```
public class Player {
  public List<Card> handCards = new List<Card>();
  public List<Card> arsenalCards = new List<Card>();
  public List<Card> ringsideCards = new List<Card>();
  public List<Card> ringAreaCards = new List<Card>();
  /* ... */
  public void DrawCardsfromArsenal(int n) {
    for (int i = 0; i < n; i++) {
      //Obtiene la última carta
      Card cardToTransfer = arsenalCards.Last();
      //Añade la carta a handCards
      handCards.Add(cardToTransfer);
      //Elimina la última carta de arsenalCards
      arsenalCards.RemoveAt(arsenalCards.Count - 1);
    }
  }
}
```

# Analizando Entregas del Proyecto

Si encapsulamos la lista de cartas en un `CardCollection`.

```
1  public class CardCollection{
2    private List<Card> _cards = new List<Card>();
3
4    public void Add(Card card)
5      => _cards.Add(card);
6
7    public void GiveTopCardTo(CardCollection destination)
8      => GiveCardTo(GetTopCard(), destination);
9
10   private Card GetTopCard()
11     => _cards.Last();
12
13   private void GiveCardTo(Card card, CardCollection destination) {
14     destination.Add(card);
15     _cards.Remove(card);
16   }
17 }
```

# Analizando Entregas del Proyecto

El código del `Player` se simplifica.

```
public class Player {
  private CardCollection handCards = new CardCollection();
  private CardCollection arsenalCards = new CardCollection();
  private CardCollection ringsideCards = new CardCollection();
  private CardCollection ringAreaCards = new CardCollection();
  /* ... */
  public void DrawCardsfromArsenal(int n)
  {
    for (int i = 0; i < n; i++)
      arsenalCards.GiveTopCardTo(handCards);
  }
}
```

# Analizando Entregas del Proyecto

**Error típico**: `04-NoEffects-Tests/3.txt`

```
813 ------------------------------------------
814 HHH: 31F, tiene 5 cartas en la mano y 2 en el arsenal.
815 HHH: 47F, tiene 0 cartas en la mano y 18 en el arsenal.
816 ------------------------------------------
817 1- Ver cartas
818 2- Jugar carta
819 3- Terminar turno
820 4- Rendirse
821 (Ingresa un número entre 1 y 4)
822 INPUT: 2
823 ------------------------------------------
824 0- [MANEUVER] *Atomic Facebuster*. Info: 4F/6D, Maneuver, Grapple.
825 1- [MANEUVER] *Hurricanrana*. Info: 11F/10D/1SV, Maneuver, Strike.
826 2- [MANEUVER] *Hurricanrana*. Info: 11F/10D/1SV, Maneuver, Strike.
827 3- [MANEUVER] *Gut Buster*. Info: 4F/5D, Maneuver, Grapple.
828 4- [MANEUVER] *Hurricanrana*. Info: 11F/10D/1SV, Maneuver, Strike.
829 ------------------------------------------
830 (Ingresa un número entre -1 y 4)
831 INPUT: 4
```

# Analizando Entregas del Proyecto

Digamos que tenemos nuestro típico `CardInfo` para leer el `json`:

```
class CardInfo
{
  public string Title { get; set; }
  public List<string> Types { get; set; }
  public List<string> Subtypes { get; set; }
  public string Fortitude { get; set; }
  public string Damage { get; set; }
  public string StunValue { get; set; }
  public string CardEffect { get; set; }
}
```

# Analizando Entregas del Proyecto

Digamos que tenemos nuestro típico `CardInfo` para leer el `json`:

```
class CardInfo
{
  public string Title { get; set; }
  public List<string> Types { get; set; }
  public List<string> Subtypes { get; set; }
  public string Fortitude { get; set; }
  public string Damage { get; set; }
  public string StunValue { get; set; }
  public string CardEffect { get; set; }
}
```

... y el típico código para leer las cartas del json:

```
string jsonString = File.ReadAllText(fileName);
var cards = JsonSerializer.Deserialize<CardInfo[]>(jsonString);
```

# Analizando Entregas del Proyecto

Para facilitarnos la vida (y por amor propio), ponemos las cartas en un diccionario:

```
public class CardCatalog {
  private Dictionary<string, CardInfo> _cardInfo;

  public CardCatalog() {
    _cardInfo = new Dictionary<string, CardInfo>();
    foreach (var cardInfo in ReadCardInfos())
      _cardInfo[cardInfo.Title] = cardInfo;
  }

  private CardInfo[] ReadCardInfos() {
    string fileName = Configuration.PathToCardsJson;
    string jsonString = File.ReadAllText(fileName);
    return JsonSerializer.Deserialize<CardInfo[]>(jsonString);
  }

  public CardInfo GetCard(string cardTitle)
    => _cardInfo[cardTitle];
}
```

# Analizando Entregas del Proyecto

... el problema viene al leer el mazo:

```
CardCatalog catalog = new CardCatalog();
List<CardInfo> cards = new List<CardInfo>();
cards.Add(catalog.GetCard("Atomic Facebuster"));
cards.Add(catalog.GetCard("Hurricanrana"));
cards.Add(catalog.GetCard("Hurricanrana"));
cards.Add(catalog.GetCard("Gut Buster"));
cards.Add(catalog.GetCard("Hurricanrana"));
```

# Analizando Entregas del Proyecto

... el problema viene al leer el mazo:

```
CardCatalog catalog = new CardCatalog();
List<CardInfo> cards = new List<CardInfo>();
cards.Add(catalog.GetCard("Atomic Facebuster"));
cards.Add(catalog.GetCard("Hurricanrana"));
cards.Add(catalog.GetCard("Hurricanrana"));
cards.Add(catalog.GetCard("Gut Buster"));
cards.Add(catalog.GetCard("Hurricanrana"));
```

Tratemos de quitar la segunda "Hurricanrana":

```
CardInfo hurricanrana = cards[2];
cards.remove(hurricanrana); // Esto borra la primera "hurricanrana" :(
```

# Analizando Entregas del Proyecto

... el problema viene al leer el mazo:

```
CardCatalog catalog = new CardCatalog();
List<CardInfo> cards = new List<CardInfo>();
cards.Add(catalog.GetCard("Atomic Facebuster"));
cards.Add(catalog.GetCard("Hurricanrana"));
cards.Add(catalog.GetCard("Hurricanrana"));
cards.Add(catalog.GetCard("Gut Buster"));
cards.Add(catalog.GetCard("Hurricanrana"));
```

Tratemos de quitar la segunda "Hurricanrana":

```
CardInfo hurricanrana = cards[2];
cards.remove(hurricanrana); // Esto borra la primera "hurricanrana" :(
```

**Solución**: Crear copias de las cartas.

# Analizando Entregas del Proyecto

De paso creemos una clase carta... que tarde o temprano la necesitaremos:

```
class Card {
  private CardInfo _cardInfo;

  public Card(CardInfo cardInfo)
    => _cardInfo = cardInfo;

  public int GetDamage()
    => _cardInfo.Damage!="#" ? Convert.ToInt32(_cardInfo.Damage) : 0;
}
```

# Analizando Entregas del Proyecto

De paso creemos una clase carta... que tarde o temprano la necesitaremos:

```
class Card {
  private CardInfo _cardInfo;

  public Card(CardInfo cardInfo)
    => _cardInfo = cardInfo;

  public int GetDamage()
    => _cardInfo.Damage!="#" ? Convert.ToInt32(_cardInfo.Damage) : 0;
}
```

Ahora podemos hacer que el catálogo retorne copias de la carta:

```
public class CardCatalog {
  /* ... */
  public Card GetCard(string cardTitle)
    => new Card(_cardInfo[cardTitle]);
}
```

# Analizando Entregas del Proyecto

Repetimos el mismo experimentos de antes:

```
CardCatalog catalog = new CardCatalog();
List<Card> cards = new List<Card>();
cards.Add(catalog.GetCard("Atomic Facebuster"));
cards.Add(catalog.GetCard("Hurricanrana"));
cards.Add(catalog.GetCard("Hurricanrana"));
cards.Add(catalog.GetCard("Gut Buster"));
cards.Add(catalog.GetCard("Hurricanrana"));

CardInfo hurricanrana = cards[2];
cards.remove(hurricanrana); // Esto borra la segunda "hurricanrana" :)
```

# Analizando Entregas del Proyecto

Repetimos el mismo experimentos de antes:

```
CardCatalog catalog = new CardCatalog();
List<Card> cards = new List<Card>();
cards.Add(catalog.GetCard("Atomic Facebuster"));
cards.Add(catalog.GetCard("Hurricanrana"));
cards.Add(catalog.GetCard("Hurricanrana"));
cards.Add(catalog.GetCard("Gut Buster"));
cards.Add(catalog.GetCard("Hurricanrana"));

CardInfo hurricanrana = cards[2];
cards.remove(hurricanrana); // Esto borra la segunda "hurricanrana" :)
```

**Consejo**: Cuando implementen la lógica de jugar cartas NO usen `removeAt(...)` porque terminarán con códigos demasiado complicados. Mejor creen copias de cartas al leer el mazo. Luego de eso pueden usar `remove(...)` sin problemas.