

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

Clase 13: Clean Code (Cap. 8)

Francisco Gazitúa Requena (fco_gazitua_requena@uc.cl)

IIC2113 Diseño Detallado de Software

23 de Septiembre, 2025

Cuestionario cap 6 y 10

Cuestionario cap 6 y 10

- **[Cap 6]** Law of Demeter aplica para objetos, no para estructuras de datos.

Cuestionario cap 6 y 10

- **[Cap 6]** Law of Demeter aplica para objetos, no para estructuras de datos.
- **[Cap 6]** Siempre es mejor usar objetos que estructuras de datos.

Cuestionario cap 6 y 10

- **[Cap 6]** Law of Demeter aplica para objetos, no para estructuras de datos.
- **[Cap 6]** Siempre es mejor usar objetos que estructuras de datos.
- **[Cap 6]** En la entrega 1, tenía sentido usar un DTO (Data Transfer Object) para guardar la información leída de las unidades desde el json.

Cuestionario cap 6 y 10

- **[Cap 6]** Law of Demeter aplica para objetos, no para estructuras de datos.
- **[Cap 6]** Siempre es mejor usar objetos que estructuras de datos.
- **[Cap 6]** En la entrega 1, tenía sentido usar un DTO (Data Transfer Object) para guardar la información leída de las unidades desde el json.
- **[Cap 6]** Los atributos de una estructura de datos deberían ser privados.

Cuestionario cap 6 y 10

- **[Cap 10]** El capítulo critica abiertamente el uso de clases "God" debido a su gran tamaño. Si tu código ya cuenta con una clase God lo mejor es que botes tu código a la basura y partas desde cero. Pero esta vez asegurándote de partir desde un diseño que te permita resolver el problema usando clases pequeñas.

Cuestionario cap 6 y 10

- **[Cap 10]** El capítulo critica abiertamente el uso de clases "God" debido a su gran tamaño. Si tu código ya cuenta con una clase God lo mejor es que botes tu código a la basura y partas desde cero. Pero esta vez asegurándote de partir desde un diseño que te permita resolver el problema usando clases pequeñas.
- **[Cap 10]** Las clases deberían tener un número pequeño de atributos.

Cuestionario cap 6 y 10

- **[Cap 10]** El capítulo critica abiertamente el uso de clases "God" debido a su gran tamaño. Si tu código ya cuenta con una clase God lo mejor es que botes tu código a la basura y partas desde cero. Pero esta vez asegurándote de partir desde un diseño que te permita resolver el problema usando clases pequeñas.
- **[Cap 10]** Las clases deberían tener un número pequeño de atributos.
- **[Cap 10]** Por lo general, es una buena señal que los métodos de una clase utilicen todos (o casi todos) los atributos de la clase.

Cuestionario cap 6 y 10

- **[Cap 10]** El capítulo critica abiertamente el uso de clases "God" debido a su gran tamaño. Si tu código ya cuenta con una clase God lo mejor es que botes tu código a la basura y partas desde cero. Pero esta vez asegurándote de partir desde un diseño que te permita resolver el problema usando clases pequeñas.
- **[Cap 10]** Las clases deberían tener un número pequeño de atributos.
- **[Cap 10]** Por lo general, es una buena señal que los métodos de una clase utilicen todos (o casi todos) los atributos de la clase.
- **[Cap 10]** Las clases deben ser pequeñas. Para medir su tamaño contamos cuántos métodos públicos tienen.

Boundaries

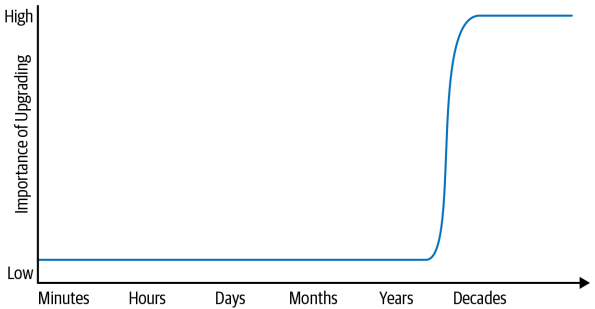
by James Grenning



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

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 equipo y decides usar un `ArrayList` de personajes.

```
1 ArrayList team = 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 equipo y decides usar un `ArrayList` de personajes.

```
1 ArrayList team = 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 equipo y decides usar un `ArrayList` de personajes.

```
1 ArrayList team = 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 `team.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.

```
2 Character unit = (Character) team[idUnit];
```

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

Boundaries

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

```
2 Character unit = (Character) team[idUnit];
```

... 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.

```
2 Character unit = (Character) team[idUnit];
```

... 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.

```
2 Character unit = (Character) team[idUnit];
```

... 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.

```
2 Character unit = (Character) team[idUnit];
```

... 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.

```
public class Team {
    private ArrayList _team = new ArrayList();

    public void AddUnit(Character unit)
        => _team.Add(unit);

    public Character GetUnit(int id)
        => (Character)_team[id];
    // ...
}
```

Boundaries

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

```
1  public class Team {
2      private ArrayList _team = new ArrayList();
3
4      public void AddUnit(Character unit)
5          => _team.Add(unit);
6
7      public Character GetUnit(int id)
8          => (Character)_team[id];
9      // ...
10 }
```

Ventajas:

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

Boundaries

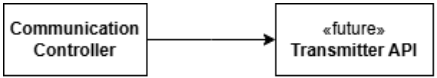
"We are not suggesting that every use of `Map` *be encapsulated in this form. Rather, we are advising you not to pass* `Maps` *(or any other interface at a boundary) around your system. If you use a boundary interface like* `Map`*, keep it inside the class, or close family of classes, where it is used.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

Boundaries

Cuando trabajemos con varios equipos, muchas veces vamos a depender de código que aún no existe

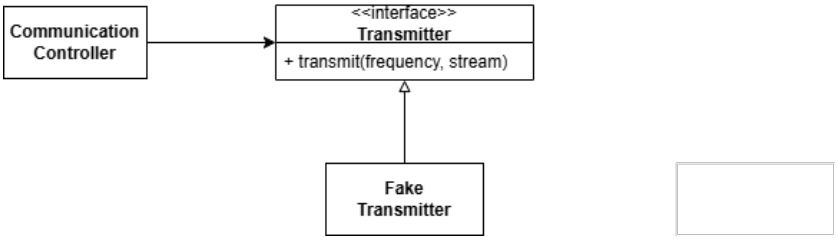
Boundaries

Cuando trabajemos con varios equipos, muchas veces vamos a depender de código que aún no existe



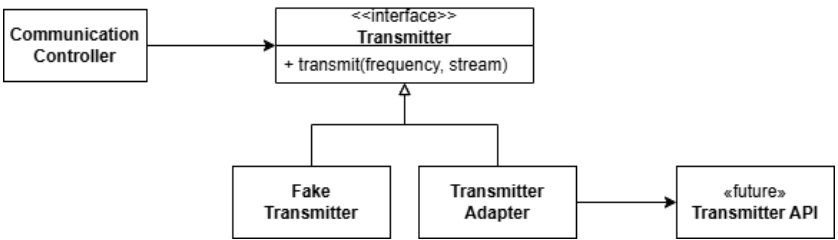
Boundaries

Cuando trabajemos con varios equipos, muchas veces vamos a depender de código que aún no existe



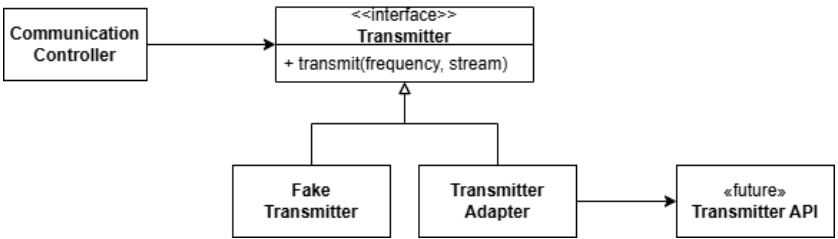
Boundaries

Cuando trabajemos con varios equipos, muchas veces vamos a depender de código que aún no existe



Boundaries

Cuando trabajemos con varios equipos, muchas veces vamos a depender de código que aún no existe



De esta forma, podemos avanzar rápidamente mientras esperamos al otro equipo

Boundaries

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

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`.

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);
```

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);
```

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

Boundaries

Ahora intentamos editar un poco la imagen.

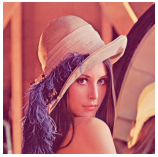
```
string imagePath = "Tests/Lenna.png";
string savePath = "Tests/Resultado.png";
Image<Rgb24> image = Image.Load<Rgb24>(imagePath);
image[20, 30] = new Rgb24(0, 255, 0);
image[21, 30] = new Rgb24(0, 255, 0);
image[20, 31] = new Rgb24(0, 255, 0);
image[21, 31] = new Rgb24(0, 255, 0);
image.Save(savePath);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

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:

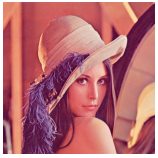
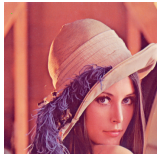


Boundaries

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:

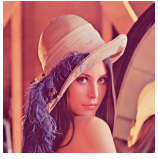
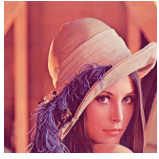
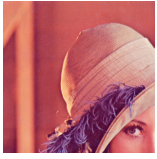



Boundaries

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:

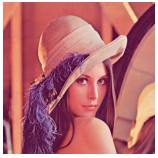
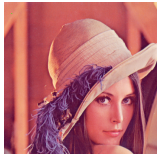
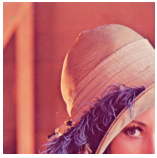
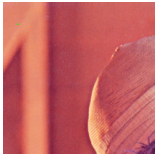




Boundaries

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:

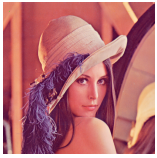
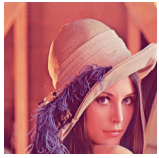
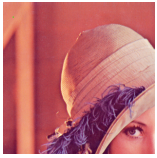
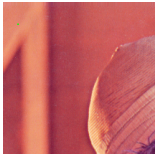





Boundaries

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:

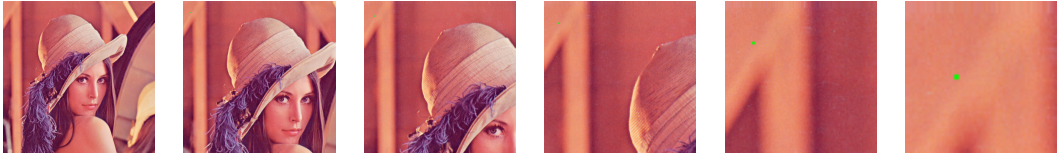






Boundaries

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

Boundaries

Ya, pero aquí viene la observación brillante:

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);
```

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.

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 }
```

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 }
```

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

Boundaries

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."

Boundaries

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.*"

¿Cuáles son los descuentos típicos en el proyecto debido al cap. 8?

Boundaries

La lección básica del capítulo es que pienses a futuro y encapsules la interacción con interfaces externas (incluyendo List, Dictionary, etc).

Boundaries

La lección básica del capítulo es que pienses a futuro y encapsules la interacción con interfaces externas (incluyendo List, Dictionary, etc).

Ejemplo: No pases `List<Unit>` por todo tu programa. Crea una clase `Team` con métodos de alto nivel para interactuar con `List<Unit>`. Luego el objeto de tipo `Team` lo puedes dar a todos lados. Si a futuro cambia la interfaz de `List` o decidimos que usar un `Dictionary` es más eficiente, entonces solo tenemos que cambiar una clase :)

Boundaries

La lección básica del capítulo es que pienses a futuro y encapsules la interacción con interfaces externas (incluyendo List, Dictionary, etc).

Ejemplo: No pases `List<Unit>` por todo tu programa. Crea una clase `Team` con métodos de alto nivel para interactuar con `List<Unit>`. Luego el objeto de tipo `Team` lo puedes dar a todos lados. Si a futuro cambia la interfaz de `List` o decidimos que usar un `Dictionary` es más eficiente, entonces solo tenemos que cambiar una clase :)

Descuentos más comunes:

- Por cada método que utilice una clase del namespace Collections (List, Dict, etc.) u otras librerías externas, a no ser que este método se encuentre en una clase cuya responsabilidad sea encapsular dicha colección.
- Por cada clase que herede de una clase externa, ya que de este modo no se encapsula ningún método de dicha clase.
- En general, no se consideran descuentos por Arrays.

Actividad Domino