Grupo: Patatas con sueño

Nayeri Azofeifa Porras. B90841

Leonel Campos Murillo. B91545

Wendy Ortiz Alfaro. B75584.

Ronald Mauricio Palma Villegas. B95811

Jeremy Vargas Artavia. B98157

## **Práctica 4: Clean Code**

**2. Investigue que es el término de code review**

Según *Code Reviews - Práctica del Desarrollo de Software*, Code Review es una práctica sistemática de desarrollo por la cual el código y sus cambios son revisados por pares, a fin de encontrar problemas, errores y vulnerabilidades.

Sin embargo en los equipos ágiles, cuando se termina de trabajar en una incidencia se revisa el código de otro desarrollador, esta debe ser una revisión circular, para que todos conozcan la base del proyecto o simplemente conocer el código y no precisamente tiene que ser durante el desarrollo.

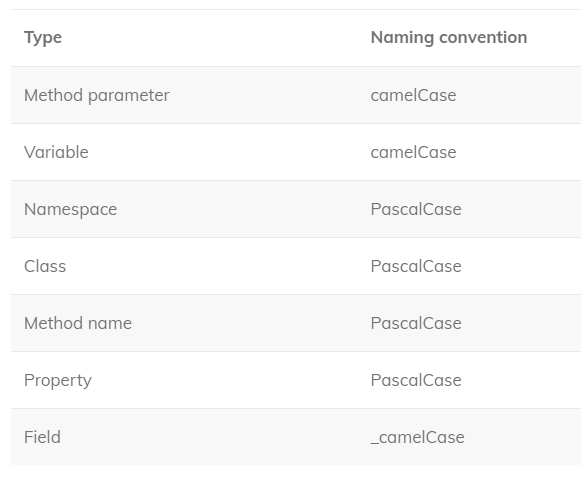
Ventajas para los equipos ágiles:

1. En el caso de los equipos ágiles que trabajan de forma descentralizada el code review hace que los integrantes del equipo puedan tener una mejor transmisión del conocimiento respecto a la base del código de todo el equipo (Atlassian, 2022).
2. Se logran mejores estimaciones luego de que cada integrante del equipo conoce las distintas partes del código (Atlassian, 2022).
3. Si se integran nuevas personas al equipo ágil por medio del code review pueden aprender y conocer la base del proyecto (Atlassian, 2022).

**3. Investigue sobre buenas prácticas de crear código limpio en C# y cualquier framework que deseen usar.**

Algunas de las buenas prácticas que se utilizan para crear código limpio en C#, se listan más adelante:

1. Convención de nomenclatura adecuada:



**Fuente:** (Kralj, 2022)

1. No usar demasiados comentarios: Un código limpio que ha sido elaborado siguiendo una serie de constructos y buenas prácticas no necesita de comentarios que expliquen la funcionalidad de este. Los comentarios solo deben ser utilizados en aquellas situaciones en donde por la naturaleza del código no es posible explicar completamente el funcionamiento con tan solo el nombre de las variables y métodos (Martin, 2009). Esto es importante puesto que un código con muchos comentarios puede tener un efecto negativo al dar una vista de un proyecto sobrecargado de información.
2. Mantener el código dentro de carpetas ordenadas. Para poder tener un mayor control sobre el proyecto que está siendo creado, es de suma importancia mantener archivos que cumplen roles diferentes, en carpetas diferentes identificadas de forma única con un nombre significativo y apropiado a la situación. De esta forma, es posible identificar rápidamente los diferentes módulos que componen el proyecto, esto permite depurar las secciones independientes del proyecto sin que los demás módulos se vean afectados (Martin, 2009).
3. Evitar la lógica condicional complicada, en lugar de crear un “switch” o múltiples ifs, se recomienda escribir diferentes funciones que respondan a esos casos. Esto promueve la legibilidad del código (Martin, 2009).
4. Construir solo lo necesario, es decir, evitar funciones no utilizadas, extras o sin ningún propósito dentro del código, esto para mejorar la compresión del código sin ninguna interrupción (Martin, 2009).

**4. También puede investigar sobre checklist que se pueden aplicar durante el code review.**

Una lista de comprobación de revisión de código contiene cosas que debe comprobar antes de confirmar o fusionar el código. El propósito de una lista de comprobación de la revisión del código es garantizar que no se olvida nada en el proceso de revisión del código. Una lista de comprobación de revisión de código eficaz no será demasiado larga, ya que eso haría que los desarrolladores fueran reacios a utilizarla. En una organización o en un equipo, es posible que quieras tener una lista de verificación de revisión de código estandarizada para mantener un alto estándar de código y cohesión en toda tu base de código (Dechalert, 2022).

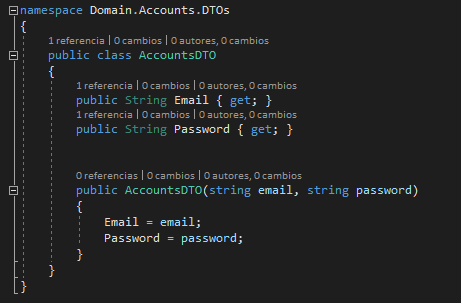
Según *Code Review Checklist: Why You Need It & How To Use It Effectively*. Los elementos a continuación son ejemplos que se utilizan para hacer la lista de verificación:

* Legibilidad y Mantenibilidad: Diferentes desarrolladores tienen métodos variados para trabajar en el código hacia un objetivo. Este método de desarrollo puede generar código ilegible e inmanejable. Una vez que el código funcione, debe volver a evaluarlo para ver si cumple con un buen estándar. Puede ser prudente volver a escribir el código, o puede que solo necesite algunos cambios de formato, documentación en línea y legibilidad.
* Rendimiento y bases de datos: Si bien la creación rápida de prototipos alienta a los desarrolladores a escribir código rápido y sucio y a preocuparse por el rendimiento cuando sea necesario, algunas aplicaciones dependen del rendimiento para brindar una buena experiencia de usuario. Si su aplicación requiere un buen rendimiento, asegúrese de tener varios elementos en su lista de verificación para abordarlo.

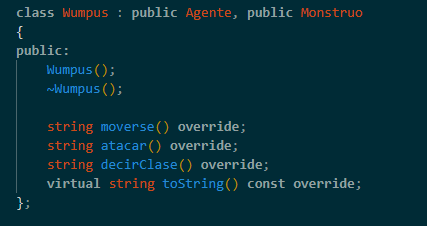
**5.Creen un checklist con los lineamientos que el equipo va a seguir para generar código limpio en el proyecto de PI. Este checklist debe tener al menos 15 reglas con sus respectivos ejemplos de que se hace y que no se hace**

* Todo el código debe ir en un mismo idioma (inglés)

Ejemplo correcto:



Ejemplo incorrecto:



* **Utilizar nombres significativos**

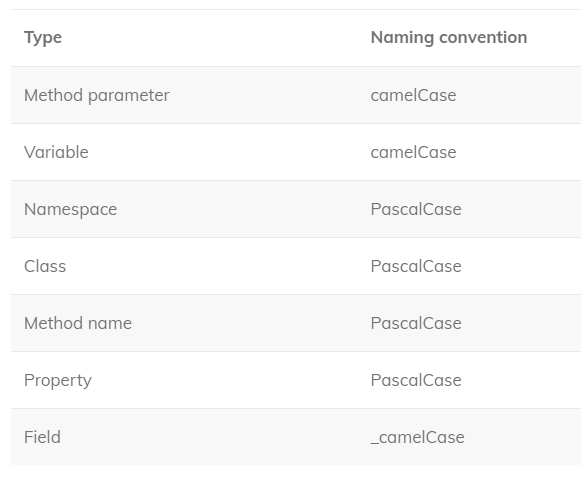
Ejemplo correcto:

if(items < MAXIMUM\_ITEM\_COUNT)

Ejemplo incorrecto:

if(i < MAXIMUM\_ITEM\_COUNT)

* **Nomenclatura:**



Ejemplo correcto:

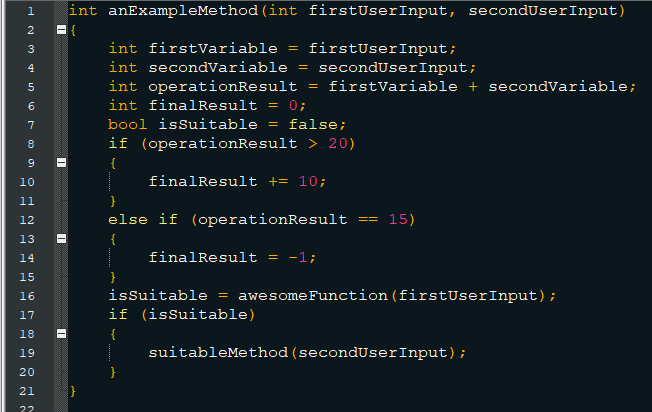
public interface IAccountRepository : IRepository<Account>

Ejemplo incorrecto:

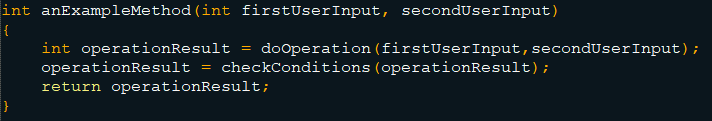
public interface Iaccountrepository : Irepository<Account>

* Métodos pequeños, máximo 20 líneas de código por método.

Ejemplo incorrecto:

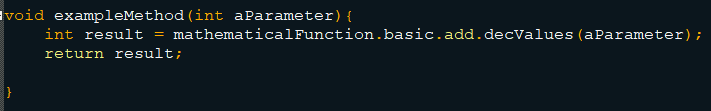


Ejemplo correcto:

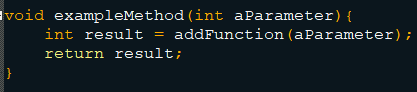


* Ley de Demeter. Un método solo debe de ser capaz de interactuar con los colaboradores inmediatos, NO con los colaboradores de sus colaboradores.

Ejemplo incorrecto:

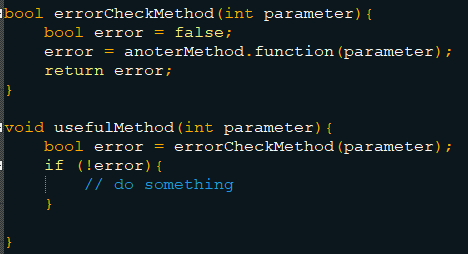


Ejemplo correcto:

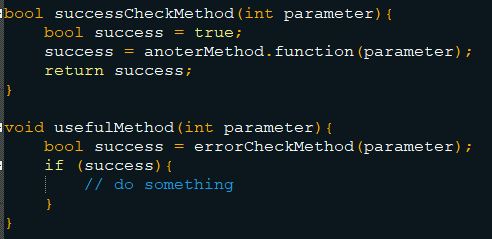


* No utilizar condicionales negativos a menos que sea estrictamente necesario

Ejemplo Incorrecto:



Ejemplo correcto:



* **Para c# Normalmente no se usan muchos comentarios por lo que si se van a usar deben ser para incluir tareas por hacer, que sean realmente realizables antes de integrar al código principal.**

**Ejemplo correcto:**

// TODO Add form validation to empty spaces

**Ejemplo Incorrecto:**

**/**/ This method renders the view and places some

// elements in the view for the user. This helps the

// user to see and do different things.

* **Evitar el código redundante.**

**Ejemplo Correcto:**

private void Print(argument)

{

Console.WriteLine(argument);

}

**Ejemplo Incorrecto:**

**Método #1:**

private void Print()

{

Console.WriteLine(name);

}

**Método** **#2:**

private void Print()

{

Console.WriteLine(lastname);

}

* **Usar llaves en una misma indentación.**

**Ejemplo correcto:**

private void Print()

{

Console.WriteLine(name);

}

**Ejemplo Incorrecto:**

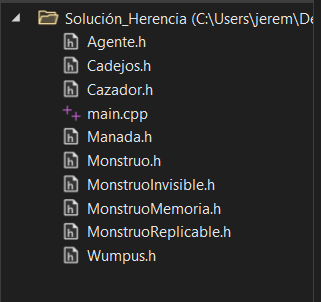
private void Print() {

Console.WriteLine(lastname);

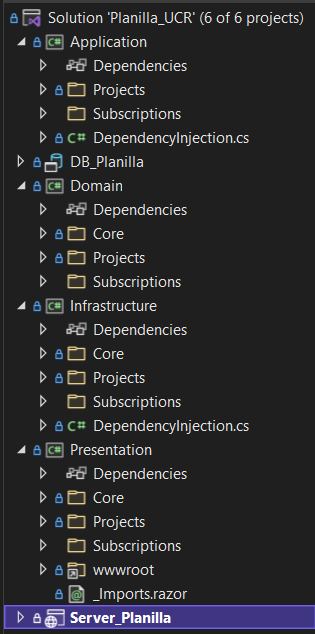
}

* **Mantener el código organizado, esto se puede lograr creando una una estructura de carpetas limpia y ordenada, para así cuando se esté buscando un código en específico se pueda encontrar de manera rápida.**

Ejemplo Incorrecto:



Ejemplo correcto:



* **Los métodos no deben de tener muchos parámetros, a lo mucho 3.**

Ejemplo incorrecto:

public void SaveHomeAddress(string name, string homeAddress, string country, string email, string fileLocation)

Ejemplo correcto:

public void SaveHomeAddress(AddressDetails addressDetails)

* **Usar constantes en vez de números mágicos. A excepción de 1 y 0.**

Ejemplo incorrecto:

if(items < 5)

Ejemplo correcto:

if(items < MAXIMUM\_ITEM\_COUNT)

* **Los nombres de las interfaces empiezan con I**

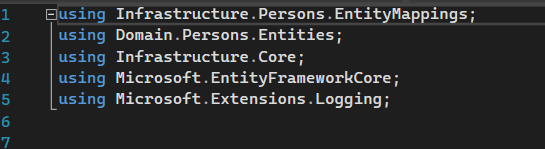
correcto:

IPersonRepository

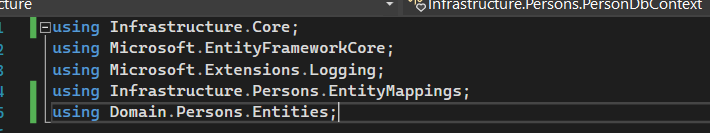
incorrecto:

* InterfacePersonRepository
* PersonRepositryInterface
* **Los using Namespace (desarrollados por nosotros) se colocan en la parte superior, antes de cualquier namespace (propio del lenguaje).**

correcto:

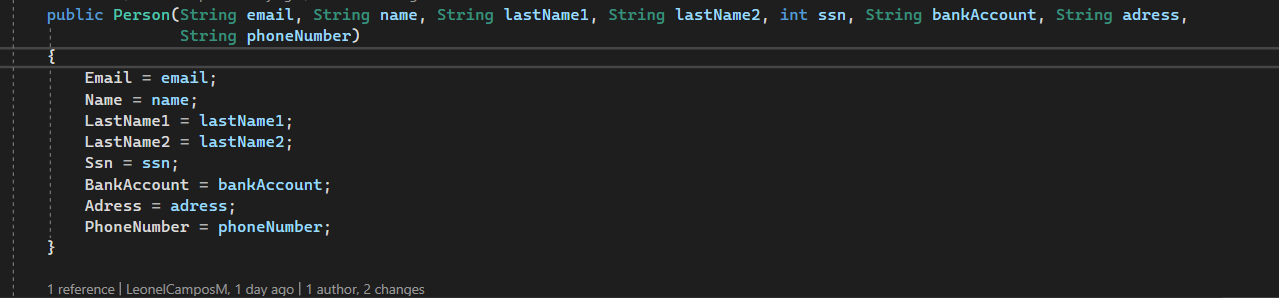


incorrecto:

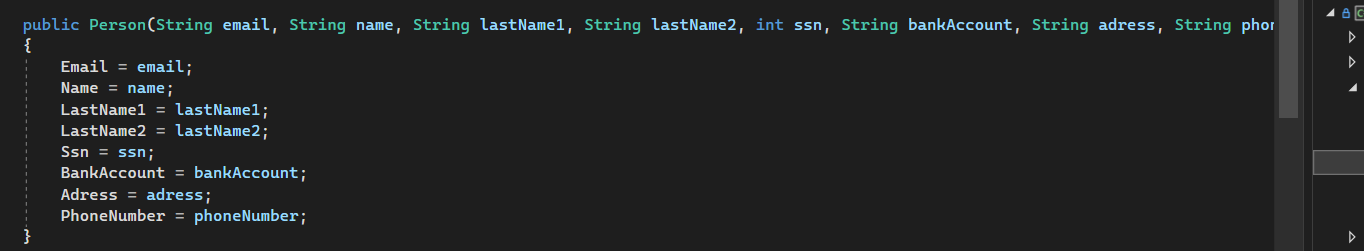


* **Si los argumentos de una función no caben todos en una línea (caracteres), deben dividirse en varias líneas, con cada línea subsiguiente alineada con el primer argumento.**

correcto:



incorrecto:



**Bibliografía**

Atlassian. (2022)*. La importancia de la revisión del código (y cómo ahorra tiempo).* <https://www.atlassian.com/es/agile/software-development/code-reviews>

Code Reviews - Práctica del Desarrollo de Software. (*s. f*.). *Práctica del Desarrollo del Software.* <https://sites.google.com/site/practicadesarrollosoft/temario/code-reviews>

Dechalert, A. (2022, 2 febrero). *Code Review Checklist: Why You Need It & How To Use It Effectively*. The Official Tabnine Blog. de <https://www.tabnine.com/blog/code-review-checklist/?utm_source=rss&utm_medium=rss&utm_campaign=code-review-checklist>

Kralj, K. (2022, 8 abril). *31 Best Clean Code C# Tips of 2022*. MethodPoet. <https://methodpoet.com/clean-code/#What_is_clean_code_in_C>

Martin, R. C.(2009). *Clean code: A handbook of agile software craftsmanship.* Upper Saddle River, NJ: Prentice Hall.