



10 Pines

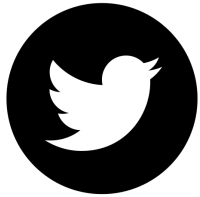
Diseño a la Gorra
Temporada 03 - Episodio 08



Hernán Wilkinson



hernan.wilkinson@10pines.com



@HernanWilkinson



<https://alagorra.10pines.com>

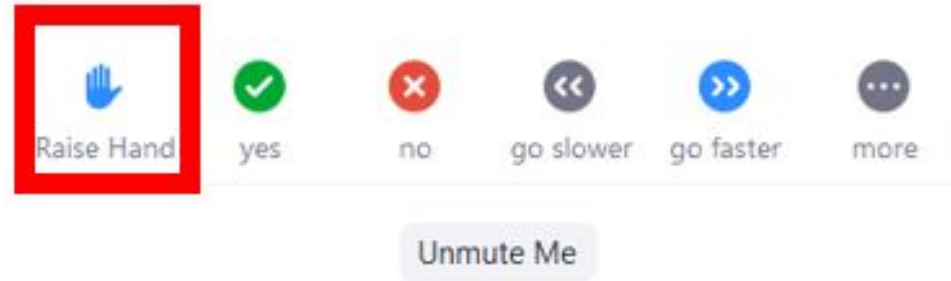


#diseño-a-la-gorra en Academia-10Pines
(si no recibieron invitación avisen!)





Estar muteados a menos que sea necesario



No voy a poder leer el chat



La comunicación visual es importante. Usarla a discreción





Diseño ¡a la gorra!

¡Bienvenidos a este exitoso Webinar que se llama "*Diseño a la Gorra*"!

¡¡El 8 de Marzo del 2023 empieza la 3er Temporada de "Diseño a la Gorra"!!

Nos vamos a encontrar todos los miércoles a las 19 hrs GMT-3, y trataremos de limitar la duración de cada episodio a una hora.

Vamos a hacer ejercicios de **TDD**, de desarrollo **iterativo/incremental**, vamos a discutir sobre **lenguajes de programación** y sus sistemas de tipos, ver cómo trabajar con código legacy y mucho más.

¡No dejes de participar! Para hacerlo inscribite [acá](#)

Durante **el año 2020** hicimos **17 episodios** donde exploramos qué es diseñar, cómo hacerlo mejor, cómo desarrollar un sistema haciendo TDD, etc.

En **el año 2022** hicimos **10 episodios** en los que tocamos los errores comunes de los lenguajes de programación, meta-programación, diseño y arquitectura.

Todo el código y presentaciones están disponibles para que lo puedan usar y consultar en cualquier momento [acá](#).

Pago voluntario



mercado
pago

\$ 2.000 - Dos 🍺

Pagar

\$ 3.000 - Una 🍕 grande

Pagar

\$ 5.000 - Un buen 🍷

Pagar

PayPal

Donate



[¿Querés abonar un monto diferente o en otra plataforma?](#)

<https://alagorra.10pines.com>

Online

Inteligencia Artificial para Ejecutivos

📅 Empieza el **16/05**

📅 Del 16/05 al 18/07. Días: Días: 9 clases a dictarse los días Martes desde el 16/5 (con excepción del 20/6 que es feriado) de 18 a 20 O sea: 16/5, 23/5, 30/5, 6/6, 13/6, 27/6, 6/7, 4/7, 11/7 y 18/7 — 18:00 a 20:00 GMT-3.

💰 **AR\$ 156.816 -** (IVA incluido) **U\$D 518 -** (impuestos incluidos)

Hacer una consulta

Inscribirme ahora **20% Dto.**



Dictado por:
Federico Lamagna

~~AR\$ 196.020-~~

AR\$ 156.816-

(IVA incluido)

~~U\$D 648-~~

U\$D 518-

(impuestos incluidos)

🔥 **Super Early Bird** 20% de descuento para los 5 primeros inscriptos hasta 21/04

🔥 **Early Bird** 10% de descuento para los segundos 5 inscriptos hasta 05/05

<http://academia.10pines.com>



iiiSorteo de remeras!!!



Principios SOLID



~~Principios~~ Heurísticas SOLID



Conclusiones del episodio anterior:

- No son principios, son Heurísticas
- El concepto de Cohesión es más básico que la heurística de Single Responsibility
 - ¿Pasará lo mismo para otros casos?
- La definición de SR
 - fue “cambiando”
 - Como “único grupo que define el cambio”, es por lo menos rara



Open/Close ~~Principle~~ Heuristic



Bertrand Meyer - OO Software Construction

The Open-Closed principle

Another requirement that any modular decomposition technique must satisfy is the Open-Closed principle:

Open-Closed principle

Modules should be both open and closed.

The contradiction between the two terms is only apparent as they correspond to goals of a different nature:

- A module is said to be open if it is still available for extension. For example, it should be possible to expand its set of operations or add fields to its data structures.
- A module is said to be closed if it is available for use by other modules. This assumes that the module has been given a well-defined, stable description (its interface in the sense of information hiding). At the implementation level, closure for a module also implies that you may compile it, perhaps store it in a library, and make it available for others (its *clients*) to use. In the case of a design or specification module, closing a module simply means having it approved by management, adding it to the project's official repository of accepted software items (often called the project *baseline*), and publishing its interface for the benefit of other module authors.



“Las entidades de software (clases, módulos, funciones, etc.) deben estar abiertas para la extensión, pero cerradas para la modificación”

- B. Martin

“... es decir, una entidad puede permitir que se amplíe su comportamiento sin modificar su código fuente.” - Wikipedia



Listing 1

Procedural Solution to the Square/Circle Problem

```
enum ShapeType {circle, square};

struct Shape
{
    ShapeType itsType;
};

struct Circle
{
    ShapeType itsType;
    double itsRadius;
    Point itsCenter;
};

struct Square
{
    ShapeType itsType;
    double itsSide;
    Point itsTopLeft;
};

//
// These functions are implemented elsewhere
//
void DrawSquare(struct Square*)
void DrawCircle(struct Circle*);
```



```
typedef struct Shape *ShapePointer;

void DrawAllShapes(ShapePointer list[], int n)
{
    int i;
    for (i=0; i<n; i++)
    {
        struct Shape* s = list[i];
        switch (s->itsType)
        {
            case square:
                DrawSquare((struct Square*)s);
                break;

            case circle:
                DrawCircle((struct Circle*)s);
                break;
        }
    }
}
```



Listing 2

OOD solution to Square/Circle problem.

```
class Shape
{
    public:
        virtual void Draw() const = 0;
};

class Square : public Shape
{
    public:
        virtual void Draw() const;
};

class Circle : public Shape
{
    public:
        virtual void Draw() const;
};

void DrawAllShapes(Set<Shape*>& list)
{
    for (Iterator<Shape*>i(list); i; i++)
        (*i)->Draw();
}
```



“Dado que el cierre no puede ser completo, debe ser estratégico. Es decir, el diseñador debe elegir los tipos de cambios contra los cuales cerrar su diseño. Esto requiere una cierta cantidad de experiencia. El diseñador experimentado conoce a los usuarios y a la industria lo suficientemente bien como para juzgar la probabilidad de diferentes tipos de cambios. Asegúrese de que se invoque el principio abierto-cerrado para los cambios más probables.

” - R. Martin



Críticas



“Este fue un sabio consejo en una época en la que el código era:

- costoso de cambiar: intente hacer un pequeño cambio y luego compilar y vincular algunos millones de líneas de C++ en la década de 1990. Esperaré.
- arriesgado cambiar, porque aún no habíamos descubierto los refactorings, menos aún los refactorings en una IDE (fuera de Smalltalk) o la programación guiada por ejemplos.
- principalmente aditivo: se escribiría un código, se lo verificaría ... y luego pasaría al siguiente archivo. Estarías traduciendo la especificación funcional detallada en código, un bloque a la vez. Cambiar el nombre de las cosas era poco común; renombrando archivos doblemente. CVS, que se convirtió en el omnipresente sistema de control de fuentes, literalmente olvidaba todo el historial de un archivo si le cambiabas el nombre, era una actividad tan poco común. Esto es fácil de pasar por alto en una era de refactorización automatizada y control de versiones basado en conjuntos de cambios.” - Dan North



“Hoy en día, el consejo equivalente si necesitas un código para hacer otra cosa es: ***¡Cambie el código para que haga otra cosa!***”

Suena trillado, pero ahora pensamos en el código como maleable como la arcilla, mientras que en los “Días Antiguos” la metáfora era más como bloques de construcción. No hubo un ciclo de retroalimentación entre la especificación y el código como lo tenemos con los tests automatizados” - Dan North



Liskov Substitution ~~Principle~~ Heuristic



“Funciones que usan punteros o referencias a clases base deben poder usar objetos de clases derivadas sin saberlo” - R. Martin



“Si para cada objeto **o1** de tipo **S**, hay un objeto **o2** de tipo **T** tal que para todos programas **P** definidos en términos de **T**, el comportamiento de **P** no cambia cuando **o1** es sustituido por **o2**, entonces **S** *es un subtipo de T.*” - B. Liskov



“Dado $P(x)$, una propiedad demostrable sobre objetos x de tipo T . Entonces $P(y)$ debería ser cierto para los objetos y de tipo S donde S es un subtipo de T ” - B. Liskov



“El principio de sustitución de Liskov (A.K.A. Diseño por Contrato) es una característica importante de todos los programas que se ajustan al principio open-closed. Solo cuando los tipos derivados son completamente sustituibles por sus tipos base, las funciones que usan esos tipos base pueden reutilizarse con impunidad, y los tipos derivados pueden cambiarse con impunidad.” - B. Martin

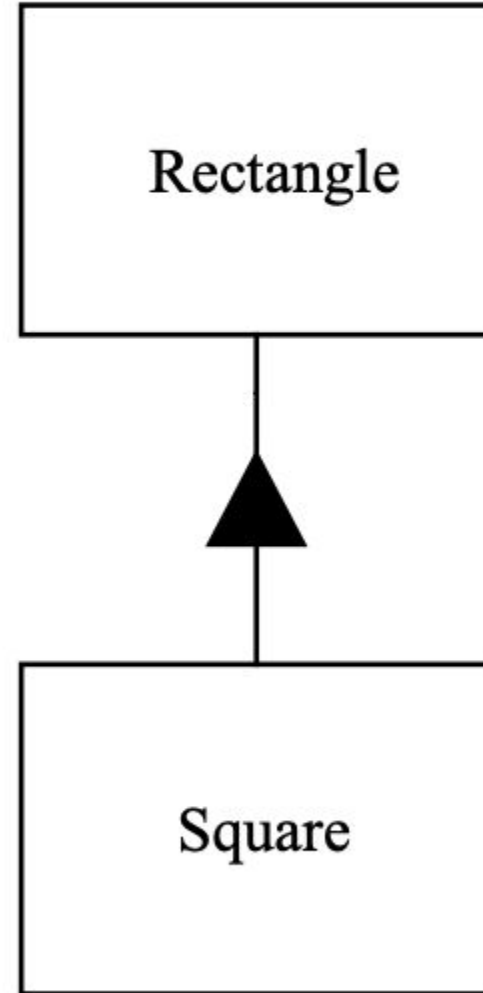


B. Meyer - Diseño por Contratos (Design By Contract)

- Preconditions cannot be strengthened in a subtype.
- Postconditions cannot be weakened in a subtype.
- Invariants of the supertype must be preserved in a subtype



Figure 1.



```
class Rectangle
{
public:
    void SetWidth(double w) {itsWidth=w;}
    void SetHeight(double h) {itsHeight=w;}
    double GetHeight() const {return itsHeight;}
    double GetWidth() const {return itsWidth;}
private:
    double itsWidth;
    double itsHeight;
};
```

```
void Square::SetWidth(double w)
{
    Rectangle::SetWidth(w);
    Rectangle::SetHeight(w);
}

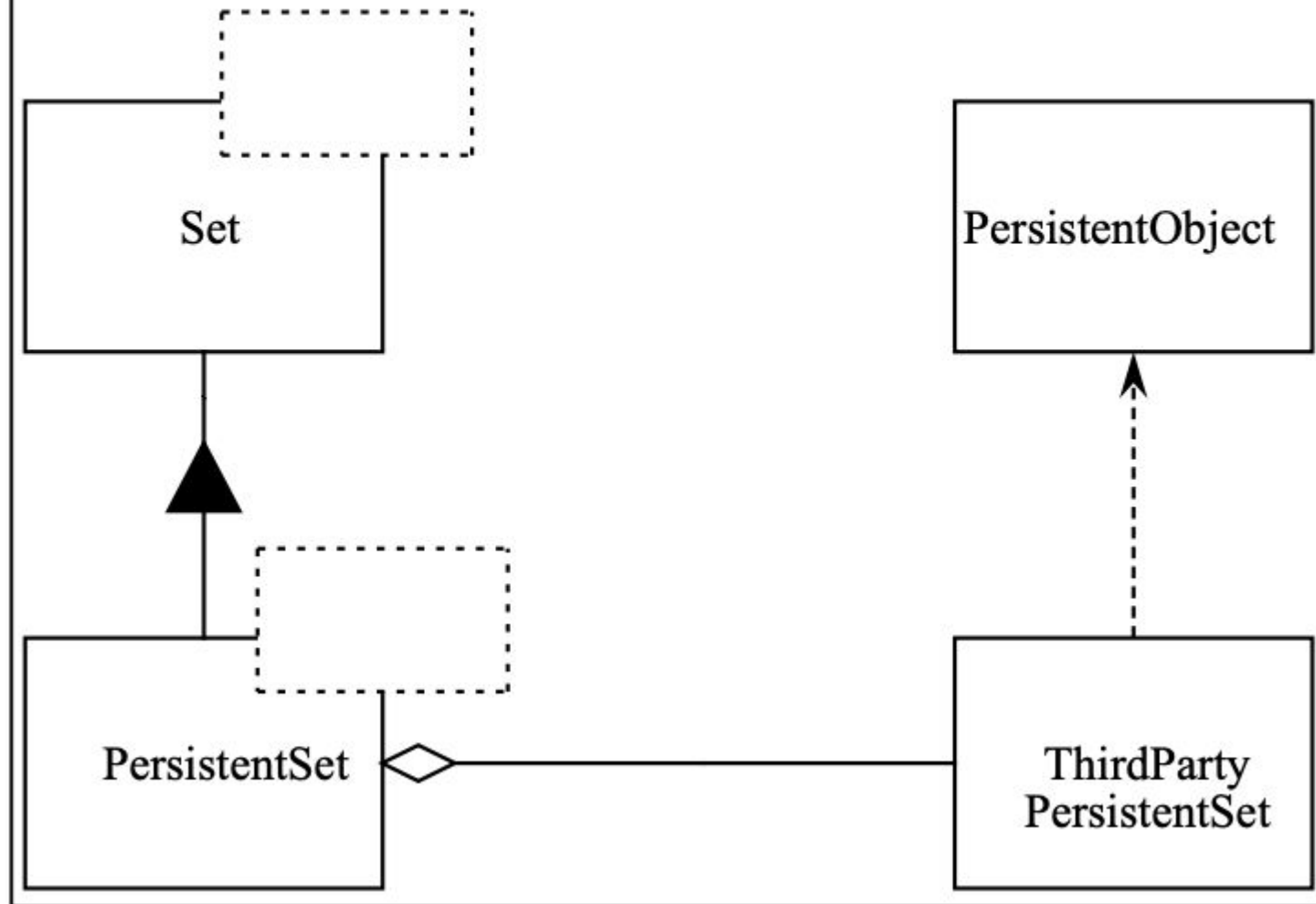
void Square::SetHeight(double h)
{
    Rectangle::SetHeight(h);
    Rectangle::SetWidth(h);
}
```



```
void g(Rectangle& r)
{
    r.SetWidth(5);
    r.SetHeight(4);
    assert(r.GetWidth() * r.GetHeight() == 20);
}
```



Figure 3. Persistent Set



Críticas



“Este es solo el **principio de menor sorpresa** aplicado a la sustitución de código y, como tal, es bastante sensato. Si te digo que algo es un subtipo válido de lo que tienes, entonces deberías poder asumir que actuará de la misma manera en cualquier sentido que te interese.

Sin embargo, el lenguaje que LSP usa para los "subtipos", junto con la forma en que la mayoría de los desarrolladores combinan los subtipos con las subclases y los caprichos de las "propiedades deseables", significa que tiende a evocar las frases de **herencia basado en "es-un" y "tiene-a"**, y su correspondiente entity-model de los años 80.” - Dan North



“En el espíritu de usar un cuchillo de mantequilla como destornillador, muchos objetos pueden "actuar como un" o "a veces-ser-usado-como-un" o "hacerse pasar-como-si" . En este contexto, lo que realmente queremos son tipos pequeños y simples que podamos componer en cualquier estructura más compleja que necesitemos, y hacer las paces con todos los matices que eso conlleva. Mi consejo, vaya sorpresa, es **"escribir código simple" sobre el que sea fácil razonar**” - Dan North



Conclusiones



Próximos Episodios

- SOLID - Parte 3 y final de 1ra Parte de la Temporada
- Refactorizando código Legacy → Para 2da Parte de la Temporada (Luego de Junio...)





Diseño ¡a la gorra!

¡Bienvenidos a este exitoso Webinar que se llama "*Diseño a la Gorra*"!

¡El 8 de Marzo del 2023 empieza la 3er Temporada de "Diseño a la Gorra"!!

Nos vamos a encontrar todos los miércoles a las 19 hrs GMT-3, y trataremos de limitar la duración de cada episodio a una hora.

Vamos a hacer ejercicios de **TDD**, de desarrollo **iterativo/incremental**, vamos a discutir sobre **lenguajes de programación** y sus sistemas de tipos, ver cómo trabajar con código legacy y mucho más.

¡No dejes de participar! Para hacerlo inscribite [acá](#)

Durante **el año 2020** hicimos **17 episodios** donde exploramos qué es diseñar, cómo hacerlo mejor, cómo desarrollar un sistema haciendo TDD, etc.

En **el año 2022** hicimos **10 episodios** en los que tocamos los errores comunes de los lenguajes de programación, metaprogramación, diseño y arquitectura.

Todo el código y presentaciones están disponibles para que lo puedan usar y consultar en cualquier momento [aquí](#)

Pago voluntario



mercado
pago

\$ 2.000 - Dos 🍺

Pagar

\$ 3.000 - Una 🍕 grande

Pagar

\$ 5.000 - Un buen 🍷

Pagar

Donate

PayPal



[¿Querés abonar un monto diferente o en otra plataforma?](#)

<https://alagorra.10pines.com>

¡Ahora sí, el sorteo!



¡Sorteo de entradas!

- Una al 50%
- Son transferibles.
- No son de uso obligatorio

Intro a Inteligencia Artificial con diseño y programación de Redes Neuronales


📅 Empieza el **17/04**

📅 Del 17/04 al 11/05. Días: 6 clases a dictarse los días Lun 17/4, Jue 20/4, Lun 24/4, Jue 4/5, Lun 8/5, Jue 11/5 — 17:00 a 20:00 GMT-3.

💰 **AR\$ 130.680 -** (IVA incluido) **USD 432 -** (impuestos incluidos)

[Hacer una consulta](#)

[Inscribirme ahora](#) **20% Dto.**

 Dictado por:
Lucas Bignone

~~AR\$ 163.350-~~
AR\$ 130.680-
(IVA incluido)
~~USD 540-~~
USD 432-
(impuestos incluidos)

🔥 **Super Early bird** 20% para los primeros 5! hasta 11/03

🔥 **Early Bird** 10% para los segundos 5! hasta 25/03



¡Sorteo de Remera!

- Lamentablemente no hay muchos talles
- Una remera distinta por semana
- Hay que buscarla o hacerse cargo del costo de envío



Muchas gracias





10 Pines

Creative Software Development



10pines.com



info@10pines.com



+54 (011) 6091-3125 / 4893-2057



Av. Leandro N. Alem 896 6° - Bs. As. - Argentina



@10pines