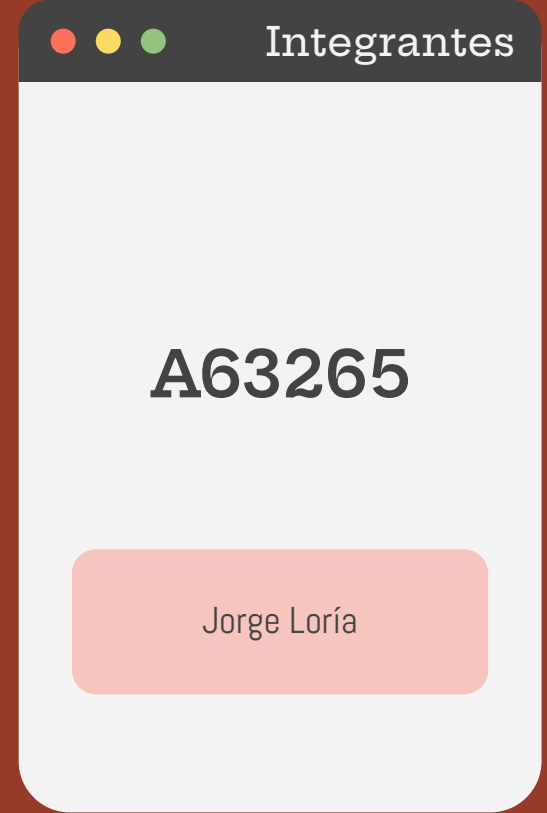
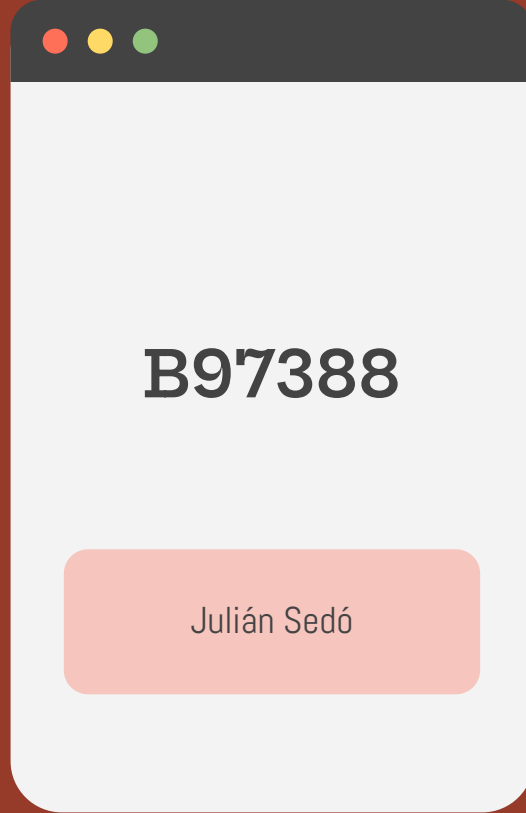
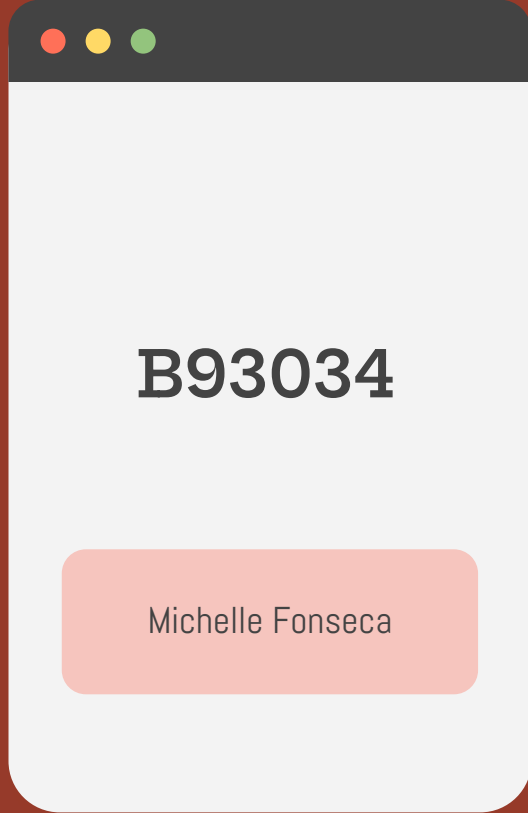


PROTOTYPE

Start now!





Introducción



Uno de los mejores analogías que explican el patrón creacional de prototype es el de la mitosis celular. La mitosis es un proceso fundamental para la vida. Durante la mitosis, una célula duplica todo su contenido, incluyendo sus cromosomas, y se divide para formar dos células hijas idénticas.



Monday

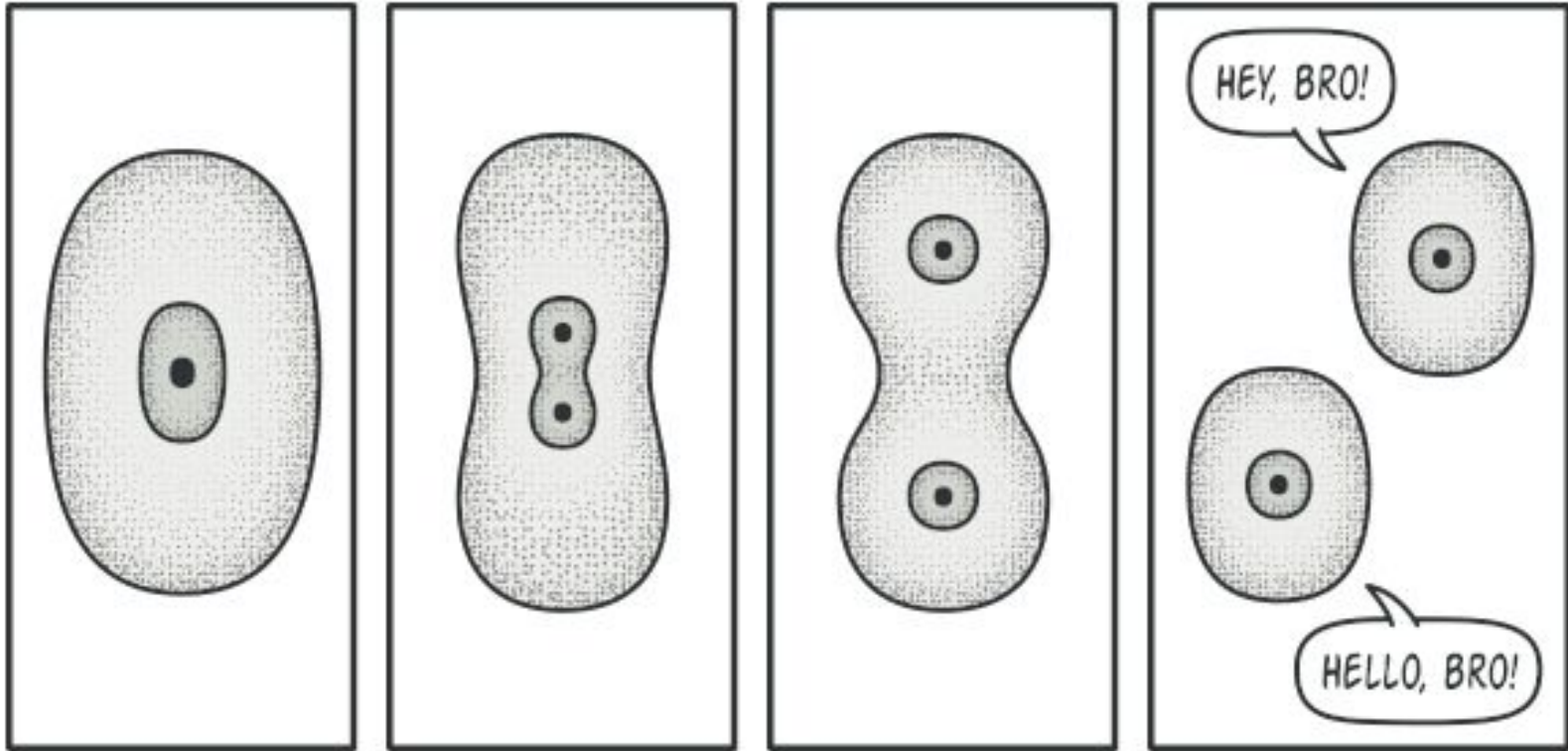
Tuesday

Wednesday

Thursday

Friday

Homework



Fuente: <https://refactoring.guru/design-patterns/prototype>



Problema



Dada la complejidad de ciertos objetos, se vuelve necesario agregar un método de clonado. De esta manera, podemos obtener una copia del objeto, sin tener que crearlo nuevamente desde cero.



Monday

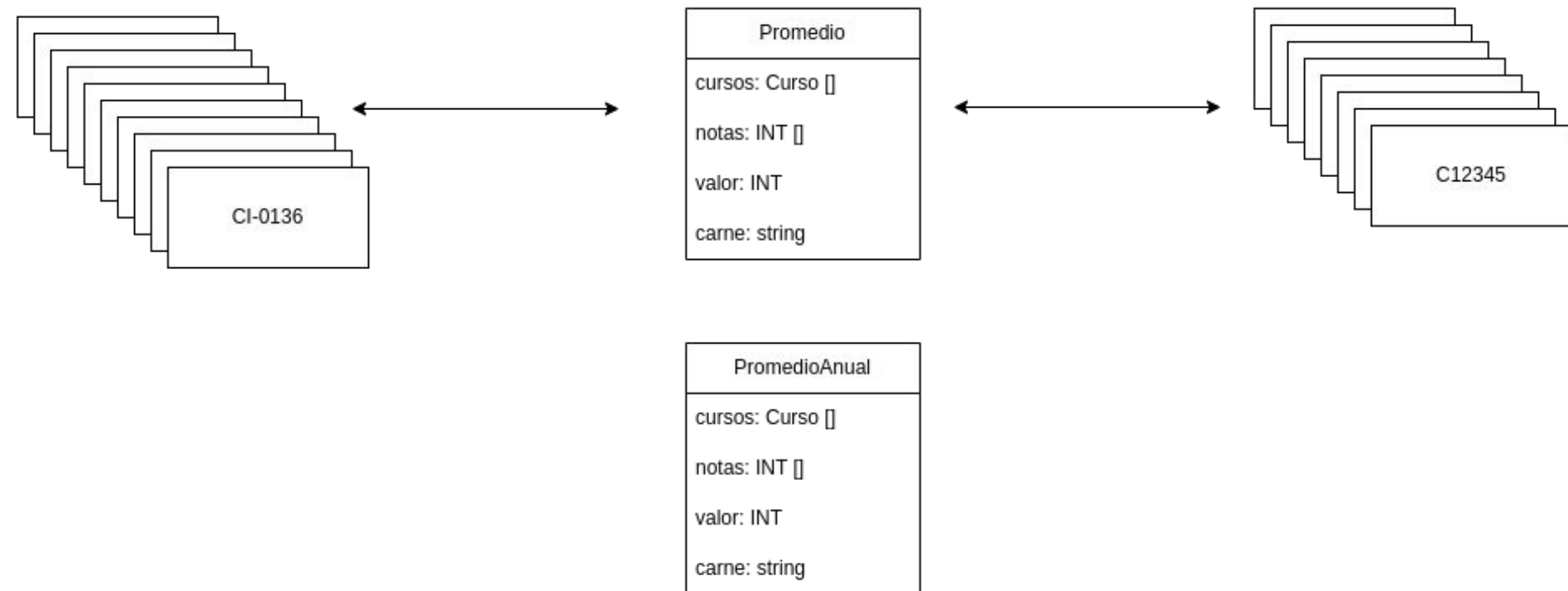
Tuesday

Wednesday

Thursday

Friday

Homework





Solución



Para evitar tener que invertir tanto tiempo para acceder a esta información, hacemos que las clases implementen una interfaz llamada Clonable, y cada vez que un estudiante solicite cualquier promedio, la primera vez que lo solicite se crea una instancia y se guarda en una estructura de datos que relacione la instancia del Promedio con una llave única



Monday

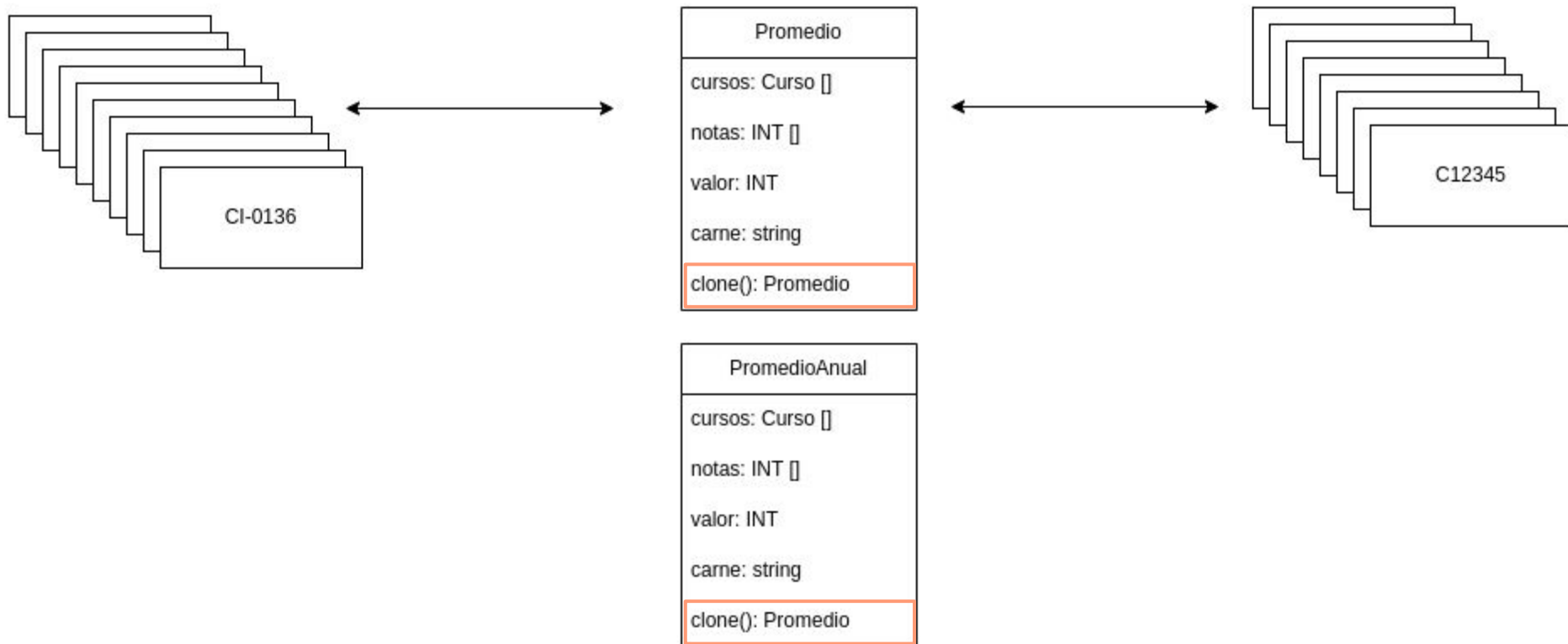
Tuesday

Wednesday

Thursday

Friday

Homework





Monday

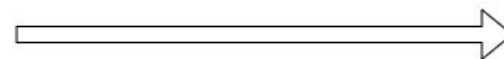
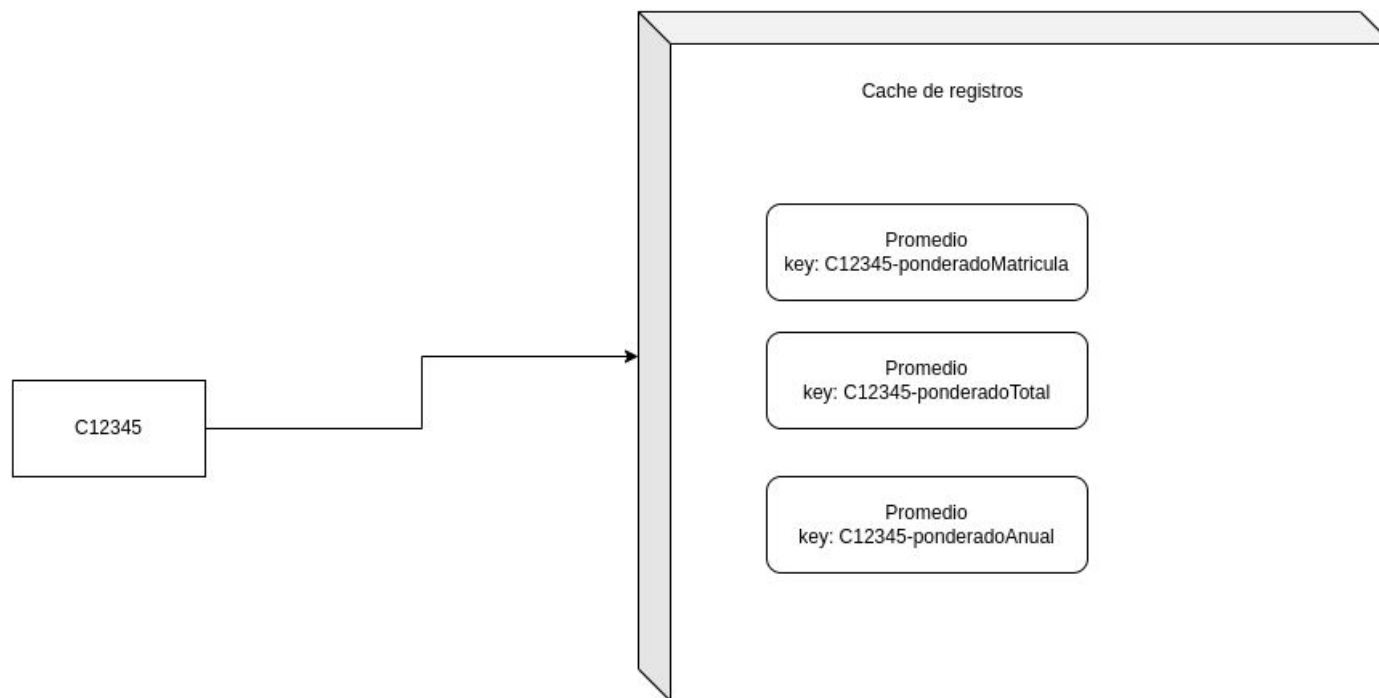
Tuesday

Wednesday

Thursday

Friday

Homework





Monday

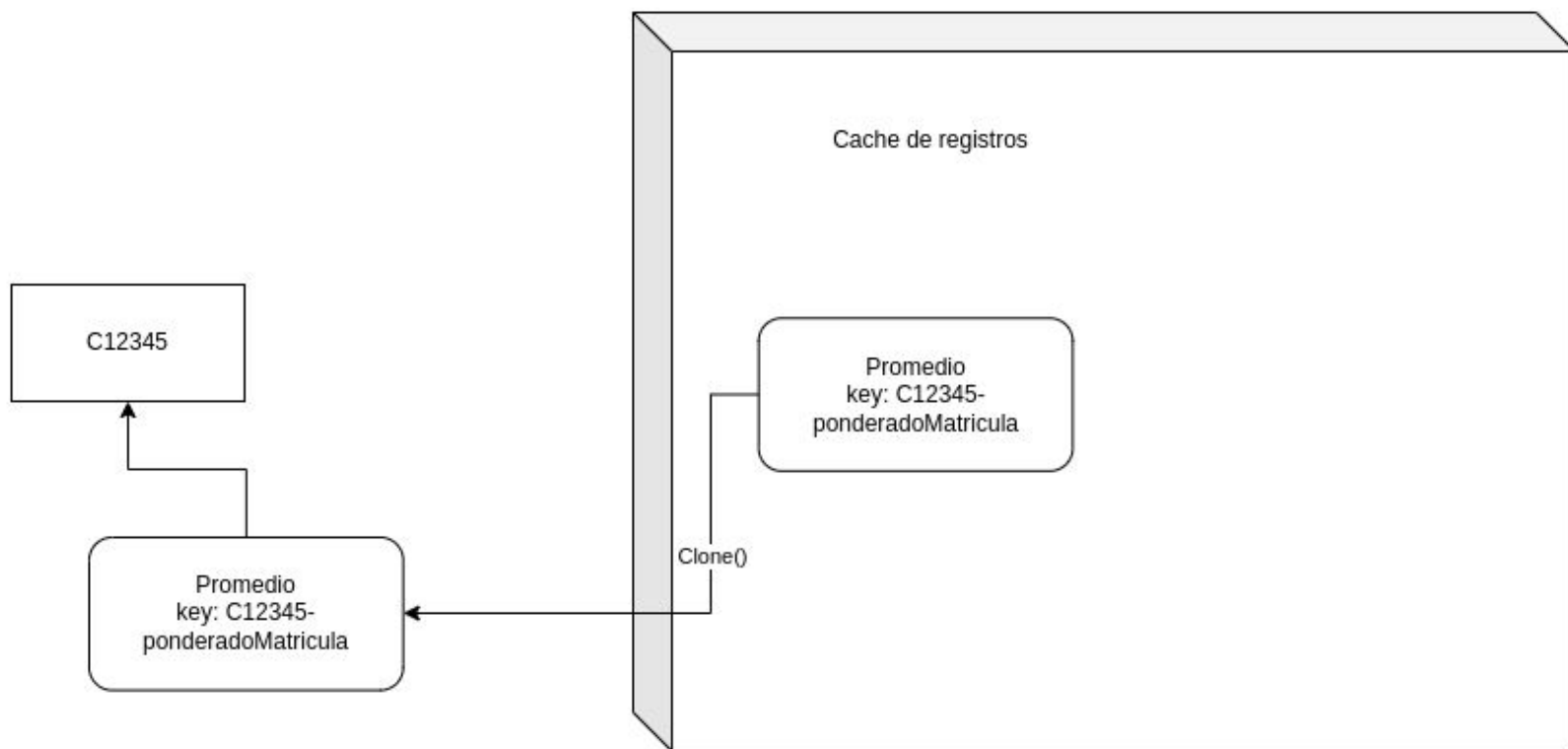
Tuesday

Wednesday

Thursday

Friday

Homework





Estructura del código



Monday

Tuesday

Wednesday


Thursday

Friday


Homework




Es uno de los patrones de diseño creacionales.



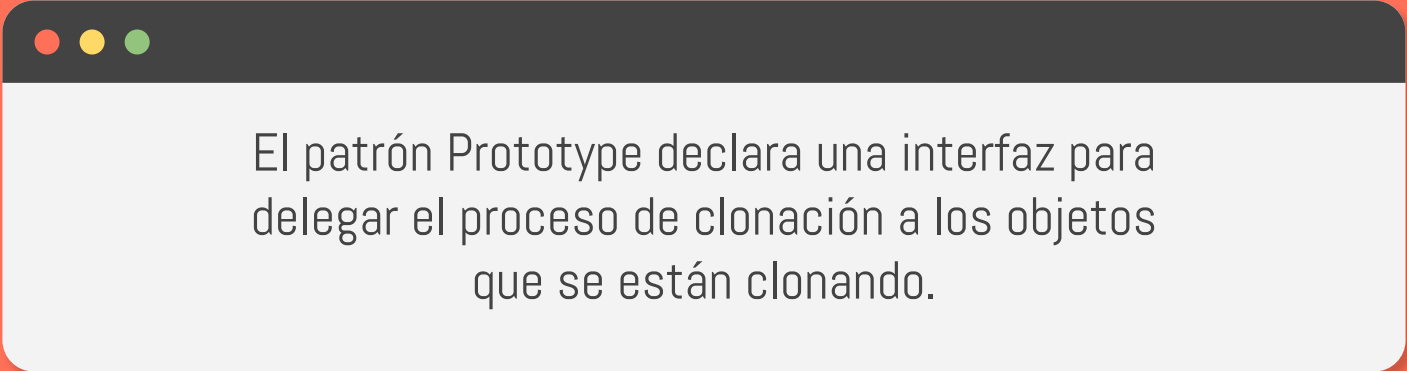
Copia un objeto existente
en lugar de crear una
nueva instancia desde
cero.



El objeto existente actúa
como prototipo y contiene
el estado del objeto



El nuevo objeto clonado
puede cambiar las
propiedades
(sólo si es necesario).

A dark gray window frame with three colored circles (red, yellow, green) on the left side.

El patrón Prototype declara una interfaz para delegar el proceso de clonación a los objetos que se están clonando.



Así es como funciona:

- Se crea un conjunto de objetos, configurados de varias maneras.
- Cuando necesite un objeto como el que ha configurado, sólo tiene que clonar un prototipo en lugar de construir un nuevo objeto desde cero.

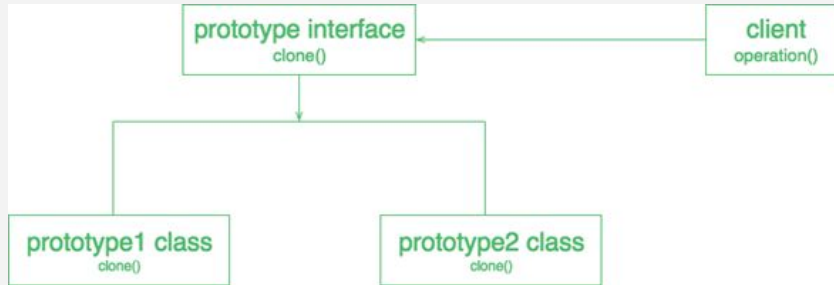


Imagen Ref: <https://www.geeksforgeeks.org/prototype-design-pattern/>

Prototipo

Registro de prototipos

Cliente



Ejemplo del código

Prototipo

```
1  abstract class Color implements Cloneable
2  {
3
4      protected String colorName;
5
6      abstract void addColor();
7
8      public Object clone()
9      {
10         Object clone = null;
11         try
12         {
13             clone = super.clone();
14         }
15         catch (CloneNotSupportedException e)
16         {
17             e.printStackTrace();
18         }
19         return clone;
20     }
21 }
```

Registro de prototipos

```
1  class blueColor extends Color
2  {
3      public blueColor()
4      {
5          this.colorName = "blue";
6      }
7
8      @Override
9      void addColor()
10     {
11         System.out.println("Blue color added");
12     }
13 }
14
15 class blackColor extends Color{
16
17     public blackColor()
18     {
19         this.colorName = "black";
20     }
21
22     @Override
23     void addColor()
24     {
25         System.out.println("Black color added");
26     }
27 }
28 }
```



Cliente



```
1 class ColorStore {  
2  
3     private static Map<String, Color> colorMap = new HashMap<String, Color>();  
  
    static  
    {  
        colorMap.put("blue", new blueColor());  
        colorMap.put("black", new blackColor());  
    }  
  
    public static Color getColor(String colorName)  
    {  
        return (Color) colorMap.get(colorName).clone();  
    }  
13  
14  
15 }
```



```
1  class Prototype
2  {
3      public static void main (String[] args)
4      {
5          ColorStore.getColor("blue").addColor();
6          ColorStore.getColor("black").addColor();
7          ColorStore.getColor("black").addColor();
8          ColorStore.getColor("blue").addColor();
9      }
10 }
```

PROBLEMS

OUTPUT

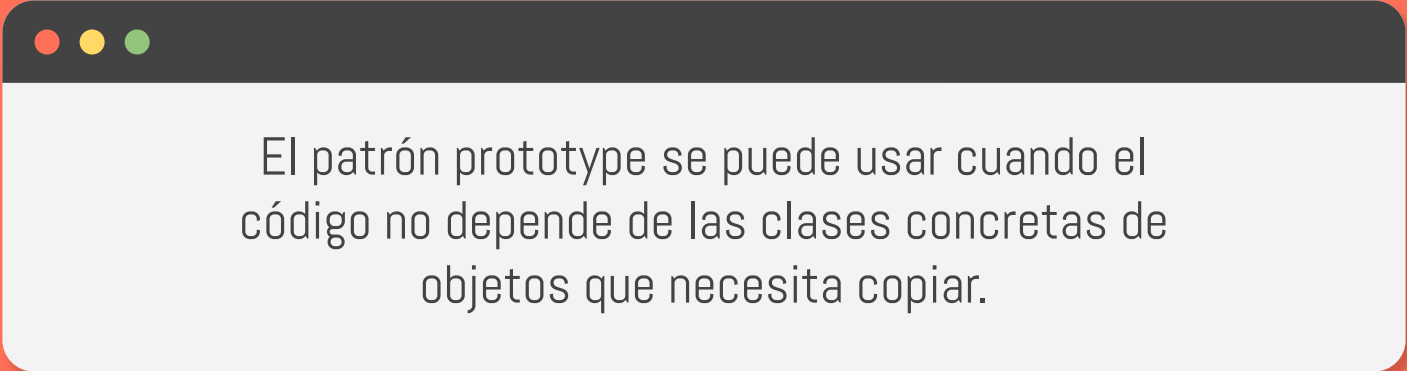
DEBUG CONSOLE

TERMINAL

```
PS C:\Users\Julián Sedó\Desktop\Diseño de software\Codigos\Prototype_gfg> java Prototype
Blue color added
Black color added
Black color added
Blue color added
PS C:\Users\Julián Sedó\Desktop\Diseño de software\Codigos\Prototype_gfg> █
```



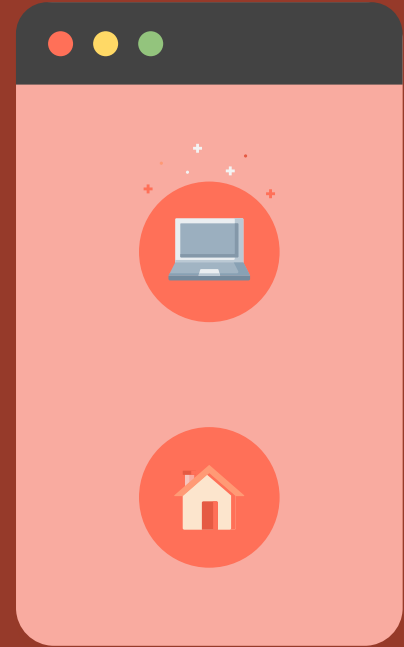

Implementación

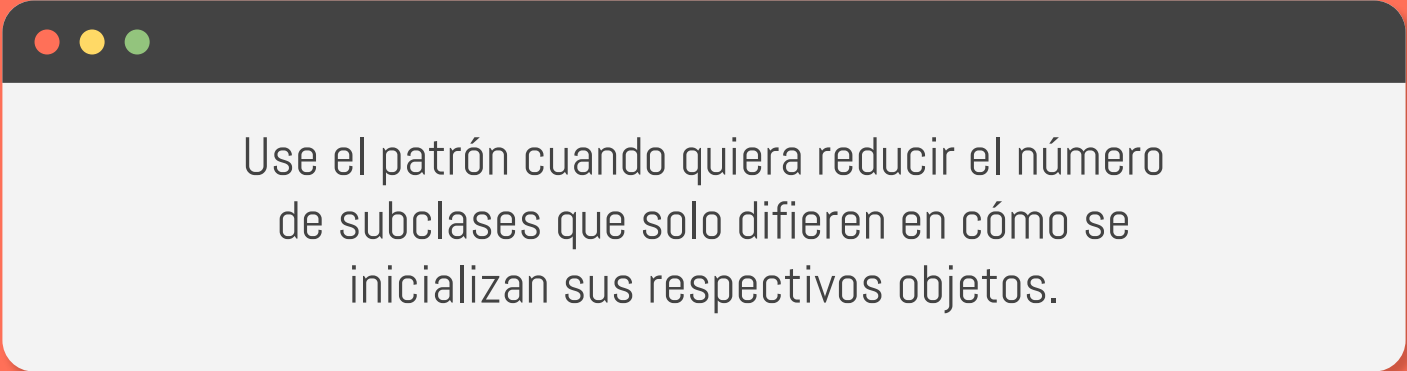
A stylized window frame with a dark gray header bar containing three colored circles (red, yellow, green) on the left. The main body of the window is white and contains the following text.

El patrón prototype se puede usar cuando el código no depende de las clases concretas de objetos que necesita copiar.

- Esto pasa cuando el código funciona con objetos que le pasan desde un código de terceros a través de una interfaz de datos, las clases concretas son desconocidas y no se puede depender de ellas

El patrón da al código una interfaz general para trabajar con todos los objetos que admiten a la clonación la cual hace que el código sea independiente de las clases concretas de un objeto que clona



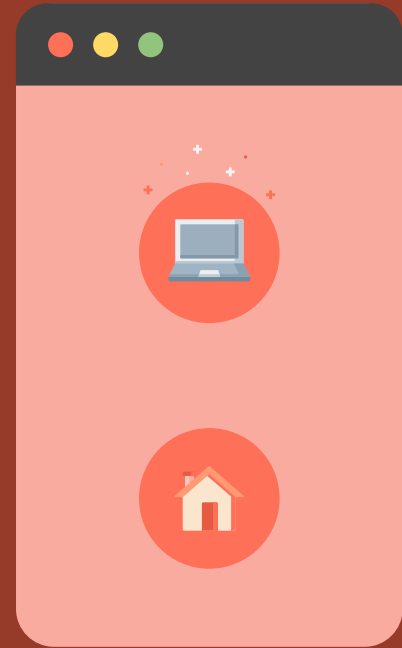
A stylized window frame with a dark gray header bar containing three colored circles (red, yellow, green) on the left. The main body of the window is white and contains the following text:

Use el patrón cuando quiera reducir el número de subclases que solo difieren en cómo se inicializan sus respectivos objetos.

- Digamos que hay una clase compleja que requiere de una configuración complicada antes de que se pueda usar y se tienen varias formas de configurar esta clase. Para reducir la duplicación se crean subclases para cada configuración.

-
- Aunque se resolvió el problema de la duplicación ahora se tienen varias subclases no significativas

El patrón permite usar un conjunto de objetos preconstruídos configurados de varias maneras como prototipos





Usar cuando:

- Las clases se instancian en tiempo de ejecución.
- El costo de crear un objeto es caro o complicado.
- Quiera mantener el número de clases en un mínimo de aplicación.
- La aplicación necesita ignorar la creación y representación de objetos.



Relación con otros patrones



01.

Muchos diseños empiezan usando **factory method** y evolucionan hacia **prototype**, entre otros.



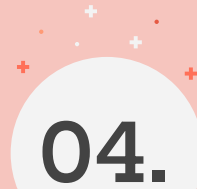
02.

Las clases de **abstract factory** se podrían hacer usando prototype para componer los métodos en estas clases



03.

Los diseños que usan **composite** y **decorator** por lo general se pueden beneficiar del uso de prototype



04.

Prototype se puede implementar como **singleton**



Consecuencias



Ventajas



Puede clonar objetos sin acoplarlos a sus clases concretas.



Puede deshacerse del código de inicialización repetido en favor de la clonación de prototipos prefabricados.



Puede producir objetos complejos de manera más conveniente



Agregar y eliminar productos en tiempo de ejecución



Desventajas



Excesivo para un proyecto que utiliza muy pocos objetos.



Cada subclase de Prototype debe implementar la operación `clone()` que puede ser difícil cuando sus partes internas incluyen objetos que no admiten la copia



Monday

Tuesday

Wednesday

Thursday

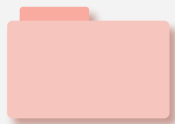
Friday

Homework

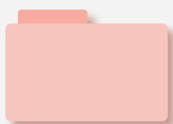
En fin



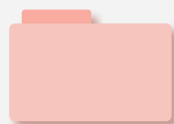
Referencias



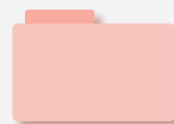
Design Patterns - Prototype Pattern. (n.d.).
https://www.tutorialspoint.com/design_pattern/prototype_pattern.htm



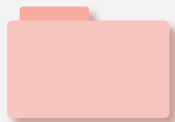
GeeksforGeeks. (2022). Prototype Design Pattern. GeeksforGeeks.
<https://www.geeksforgeeks.org/prototype-design-pattern/>



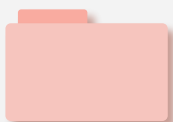
Pankaj. (2022). Prototype Design Pattern in Java. DigitalOcean.
<https://www.digitalocean.com/community/tutorials/prototype-design-pattern-in-java>



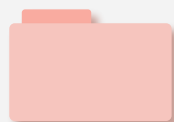
Prototype Design Pattern - Javatpoint. (n.d.).
[www.javatpoint.com.
https://www.javatpoint.com/prototype-design-pattern](https://www.javatpoint.com/prototype-design-pattern)



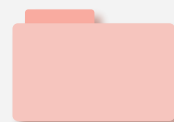
Prototype Pattern. (n.d.-a).
<https://www.patterns.dev/posts/prototype-pattern>



Prototype Pattern. (n.d.-b).
<https://www.patterns.dev/posts/prototype-pattern>



Refactoring.Guru. (2023). Prototype. Refactoring.Guru.
<https://refactoring.guru/design-patterns/prototype>



The GoF Design Patterns Memory - Learning Object-Oriented Design & Programming. (2017). w3sDesign.
<http://w3sdesign.com/?gr=c04&ugr=proble>