Aplicaciones Web

Patrones de diseño hipermedia

Sergio Luján Mora
Edgar Valarezo
¿Qué es un patrón de diseño?
A Pattern Language: Towns, Buildings, Construction

Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, Shlomo Angel

Oxford University Press, 1977
... the main outdoor spaces are given their character by site repair (104), south facing outdoors (105) and positive outdoor space (106). But you can refine them, and complete their character by making certain that every space always has a view out into some other larger one, and that all the spaces work together to form hierarchies.

* * *

Outdoors, people always try to find a spot where they can have their backs protected, looking out toward some larger opening, beyond the space immediately in front of them.

In short, people do not sit facing brick walls—they place themselves toward the view or toward whatever there is in the distance that comes nearest to a view.

Simple as this observation is, there is almost no more basic statement to make about the way people place themselves in space. And this observation has enormous implications for the spaces in which people can feel comfortable. Essentially, it means that any place where people can feel comfortable has

1. A back.
2. A view into a larger space.

In order to understand the implications of this pattern, let us look at the three major cases where it applies.

In the very smallest of outdoor spaces, in private gardens, this pattern tells you to make a corner of the space as a "back" with a seat, looking out on the garden. If it is rightly made, this corner will be snug, but not at all claustrophobic.

---

II4 HIERARCHY OF OPEN SPACE

Slightly larger in scale, there is the connection between a terrace or an outdoor room of some kind and a larger open space, the street or a square. The most common form of the pattern at this scale is the front stoop, which forms a definite enclosure and a back, off the public street.

* * *

 Terrace and street or square.

At the largest scale, this pattern tells you to open up public squares and greens, at one end, to great vistas. At this scale, the square itself acts as a kind of back which a person can occupy, and from which he can look out upon an even larger expanse.

* * *

 Square and vista.

Therefore:

Whatever space you are shaping—whether it is a garden, terrace, street, park, public outdoor room, or courtyard, make sure of two things. First, make at least one smaller space, which looks into it and forms a natural back for it. Second, place it, and its openings, so that it looks into at least one larger space.

When you have done this, every outdoor space will have
Razón:
“Outdoors, people always try to find a spot where they can have their backs protected, looking out toward some larger opening, beyond the space immediately in front of them.”

Patrón:
“Oh whatever space you are shaping – whether it is a garden, terrace, street, park, public outdoor room, or courtyard – make sure of 2 things. First, make sure at least one smaller space, which looks into it and forms a natural back for it. Second, place it, and its openings, so that is looks into at least one larger space.”
Terrace and street or square.

Seat and garden.

Square and vista.
¿Algún ejemplo de patrón de diseño en algún aspecto de la vida?
Ceviche de camarón

In English

El ceviche de camarón es uno de los ceviches ecuatorianos más populares. El ceviche, también conocido como cebiche (ambos nombres son aceptables), es un plato tradicional muy popular en todo el Ecuador, pero sobre todo en la playa. Una de las ventajas de este ceviche es que usa camarones que ya están cocidos – a diferencia del ceviche de pescado donde el pescado se “cocina” con la acidez del limón sutil. Este ceviche es ideal para los que no pueden comer mariscos crudos o quieren probar el ceviche por primera vez, pero quizás tengan un poco de miedo a la idea de preparar un ceviche en casa. Claro, que mi recomendación es que también intenten preparar el ceviche de pescado, por lo general si se prepara con pescado fresco de alta calidad y se lo mantiene en refrigeración es muy raro que le pase algo.
**Ingredientes:**

- 2 libras de camarón ya cocinado, pelados y sin venas
- 2 cebollas coloradas pequeñas, cortadas en rodajas súper finas
- 4 tomates, cortadas en rodajas finas
- El jugo de unos 15-20 limones
- El jugo de una naranja
- ½ taza de salsa de tomate
- 1 manjo de cilantro o culantro, picado finamente
- Sal y aceite

**Preparación:**

1. Lavar las rodajas de cebolla con sal y agua, luego enjuaguelas.
2. Mezcle todos los ingredientes y deje reposar en la refrigeradora por lo menos durante dos horas.
3. Sirva acompañado de aji, chifles, patacones o canguil.
Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.
Efectivo
Resuelve problemas similares en ocasiones anteriores

Reutilizable
Aplicable a diferentes problemas de diseño en distintas circunstancias
¿Qué es un Patrón de Diseño?

Personas que lo han encontrado útil: 0 de 106 - Valorar este tema

Por Nicolás Tedeschi

Contenido

¿Qué es un Patrón de Diseño?
Patrones Creacionales
Patrones Estructurales
Patrones de Comportamiento
Conclusión

¿Qué es un Patrón de Diseño?

Esta fue la primera pregunta que me hizo cuando comence a investigar sobre este tema. Al principio no tenía mucha idea de por donde comenzar, por lo que mi primera reacción fue realizar una búsqueda en Internet y obtener de esta manera alguna base sobre la cual apoyarme. La definición que más me gusto fue la siguiente:

“Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software.”

En otras palabras, brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares. Debemos tener presente los siguientes elementos de un patrón: su nombre, el problema (cuando aplicar un patrón), la solución (descripción abstracta del problema) y las consecuencias (costos y beneficios).

Grande fue mi sorpresa al averiguar que existen varios patrones de diseño popularmente conocidos, los cuales se clasifican como se muestra a continuación:

- Patrones Creacionales: inicialización y configuración de objetos.
- Patrones Estructurales: Separan la interfaz de la implementación. Se ocupan de cómo las clases y objetos se agrupan, para formar estructuras más grandes.
- Patrones de Comportamiento: Más que describir objetos o clases, describen la comunicación entre ellos.

Veamos un poco en qué consisten los distintos tipos de patrones, cuáles son sus fines y qué beneficios nos aportan.

Principio de la página

Patrones Creacionales

Fábrica Abstracta (Abstract Factory)

El problema a solucionar por este patrón es el de crear diferentes familias de objetos, como por ejemplo la creación de interfaces gráficas de distintos tipos (ventana, menu, botón, etc.).

Método de Fabricación (Factory Method)
Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software.
Brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares. Debemos tener presente los siguientes elementos de un patrón: su nombre, el problema (cuando aplicar un patrón), la solución (descripción abstracta del problema) y las consecuencias (costos y beneficios).
Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

1995
La Banda de los Cuatro
Gang of Four
Nombre del patrón
Clasificación
Intención
También conocido como
Motivación
Aplicación
Estructura
Partcipantes
Colaboradores
Consecuencias
Implementación
Ejemplo de código
Usos conocidos
Patrones relacionados
Catálogo de patrones:
Abstract Factory
Adapter
Bridge
Builder
Chain of Responsibility
Command
Composite
Decorator
Facade
Factory Method
Flyweight
Interpreter
Iterator
...

...
Intent

Ensure a class only has one instance, and provide a global point of access to it.

Motivation

It's important for some classes to have exactly one instance. Although there can be many printers in a system, there should be only one printer spooler. There should be only one file system and one window manager. A digital filter will have one A/D converter. An accounting system will be dedicated to serving one company.

How do we ensure that a class has only one instance and that the instance is easily accessible? A global variable makes an object accessible, but it doesn't keep you from instantiating multiple objects.

A better solution is to make the class itself responsible for keeping track of its sole instance. The class can ensure that no other instance can be created (by intercepting requests to create new objects), and it can provide a way to access the instance. This is the Singleton pattern.

Applicability

Use the Singleton pattern when

- there must be exactly one instance of a class, and it must be accessible to clients from a well-known access point.

- when the sole instance should be extensible by subclassing, and clients should be able to use an extended instance without modifying their code.

Structure

```
Singleton

static Instance() { return uniqueInstance; }
SingletonOperation()
GetSingletonData()

static uniqueInstance
singletonData
```

Participants

- Singleton

  - defines an Instance operation that lets clients access its unique instance. Instance is a class operation (that is, a class method in Smalltalk and a static member function in C++).
Collaborations

- Clients access a Singleton instance solely through Singleton's Instance operation.

Consequences

The Singleton pattern has several benefits:

1. *Controlled access to sole instance*. Because the Singleton class encapsulates its sole instance, it can have strict control over how and when clients access it.

2. *Reduced name space*. The Singleton pattern is an improvement over global variables. It avoids polluting the name space with global variables that store sole instances.

3. *Permits refinement of operations and representation*. The Singleton class may be subclassed, and it's easy to configure an application with an instance of this extended class. You can configure the application with an instance of the class you need at run-time.

4. *Permits a variable number of instances*. The pattern makes it easy to change your mind and allow more than one instance of the Singleton class. Moreover, you can use the same approach to control the number of instances that the application uses. Only the operation that grants access to the Singleton instance needs to change.

5. *More flexible than class operations*. Another way to package a singleton's functionality is to use class operations (that is, static member functions in C++ or class methods in Smalltalk). But both of these language techniques make it hard to change a design to allow more than one instance of a class. Moreover, static member functions in C++ are never virtual, so subclasses can't override them polymorphically.

Implementation

Here are implementation issues to consider when using the Singleton pattern:

1. *Ensuring a unique instance*. The Singleton pattern makes the sole instance a normal instance of a class, but that class is written so that only one instance can ever be created. A common way to do this is to hide the operation that creates the instance behind a class operation (that is, either a static member function or a class method) that guarantees only one instance is created. This operation has access to the variable that holds the unique instance, and it ensures the variable is initialized with the unique instance before returning its value. This approach ensures that a singleton is created and initialized before its first use.

   You can define the class operation in C++ with a static member function `instance` of the Singleton class. Singleton also defines a static member variable `_instance` that contains a pointer to its unique instance.

   The `singleton` class is declared as

   ```
   class Singleton {
   public:
   ```
static Singleton* Instance();
protected:
    Singleton();
private:
    static Singleton* _instance;
};

The corresponding implementation is

Singleton* Singleton::Instance = 0;
Singleton* Singleton::Instance () {
    if (_instance == 0) {
        _instance = new Singleton;
    }
    return _instance;
}

Clients access the singleton exclusively through the Instance member function. The variable _instance is initialized to 0, and the static member function Instance returns its value, initializing it with the unique instance if it is 0. Instance uses lazy initialization; the value it returns isn’t created and stored until it’s first accessed.

Notice that the constructor is protected. A client that tries to instantiate Singleton directly will get an error at compile-time. This ensures that only one instance can ever get created.

Moreover, since the _instance is a pointer to a Singleton object, the Instance member function can assign a pointer to a subclass of Singleton to this variable. We’ll give an example of this in the Sample Code.

There’s another thing to note about the C++ implementation. It isn’t enough to define the singleton as a global or static object and then rely on automatic initialization. There are three reasons for this:

1. We can’t guarantee that only one instance of a static object will ever be declared.
2. We might not have enough information to instantiate every singleton at static initialization time. A singleton might require values that are computed later in the program’s execution.
3. C++ doesn’t define the order in which constructors for global objects are called across translation units [ES99]. This means that no dependencies can exist between singletons; if any do, then errors are inevitable.

An added (albeit small) liability of the global/static object approach is that it forces all singletons to be created whether they are used or not. Using a static member function avoids all of these problems.

In Smalltalk, the function that returns the unique instance is implemented as a class method on the Singleton class. To ensure that only one instance is created, override the new operation. The resulting Singleton class might have the following two class methods, where SoleInstance is a class variable that is not used anywhere else:

    new
    self error: 'cannot create new object'
class Singleton {
    public:
        static Singleton* Instance();
    protected:
        Singleton();
    private:
        static Singleton* _instance;
};

Singleton* Singleton::_instance = 0;

Singleton* Singleton::Instance () {  
    if (_instance == 0) {  
        _instance = new Singleton;
    }  

    return _instance;
}
Figure K8.1

External links point to other Web sites and let people see how your Web site is connected with other people, places, and things. Some external links in this screen shot include the related stock quotes on the left, the content module on the bottom left labeled "ELSEWHERE ON THE WEB", and the link to Rite Aid under "On the Net" near the bottom.
**BACKGROUND**

It can be difficult for customers to know if a link points to something in your Web site or goes to an outside site. Including links to external Web sites can improve your credibility, as in GRASSROOTS INFORMATION SITES (A6), but it also makes it easy for customers to leave your Web site unintentionally. This pattern explains some techniques for organizing external links so that there are no surprises for your customers.

**PROBLEM**

Most sites have links to other Web sites. These external links need to be treated in a special manner so that customers understand that they lead to other Web sites that are not managed by the current Web site.

Providing external links is important for a site because it helps build trust and credibility. Visitors appreciate a site owner who is willing to reference another site to keep customers better informed and satisfied. But site owners are rightly concerned that customers will leave their sites to follow an external link. And visitors are sometimes surprised when they click on a link and unexpectedly leave a site. You can address these concerns by building in some safeguards, as described in the sections that follow.

**Consider Including Links That Lead to External Web Sites**

Help your customers by providing additional useful information that you don’t have on your site. Links might go to partner Web sites, client Web sites, press releases, articles, reviews, products, and services that complement your own.

**Minimize Link Rot**

A site operator could change the content of a page or decommission a page address, and you would never know about it until you tested your external links. This problem is often called link rot. If your site has many external links, testing the links can be time-consuming. In addition, the external site to which you link may not want you to link to it.

There are three things you can do. The first is to use tools to help check the hyperlinks on your Web site, to ensure that they’re still going to the right place. The second is to link to PERMALINKS (K15), which are links
that are intended to always work. The third is to get permission from the external site. Getting permission is useful for several reasons:

- External site operators can give you their conditions for linking to their sites, perhaps agreeing to link to you as well.
- You can inform the external site operators that your site relies on their current link structure, thereby hopefully reducing link rot.
- Talking to another site operator helps you understand how to link to other sites so that the same page addresses will contain the content you wish to reference, again reducing link rot.

Receiving permission does not guarantee that the external links will continue to work in the future, but you will be much better informed as a result.

**Set Expectations, Mark External References** — News sites that reference other news sites often collect all their external references in one clearly marked area. For example, the CNN Web site places external links at the bottom of each news article. Other news aggregators, like Yahoo!, put a marker—such as [external]—before each link to another site. These strategies inform customers that if they click the link, they will no longer be on the current site, so that they will not be surprised to be transported away. You can use similar strategies by putting external content in specific areas or clearly marking an external reference on the link itself.

**Consider Combining Embedded Links with External Links** — One technique that the New York Times uses for its news articles is to merge embedded links with external links. If customers click on an embedded link, they jump to the bottom of the page, where there is a section identified as external links.

The advantage of this approach is that it makes it easy for readers to see all of the external links after reading a news article. It also lets customers go to external Web sites, making clear which links point to those sites.

**Open New Windows for External Links** — Open a new window for an external link when you want customers to maintain the context of the original site, especially when they’re in the middle of a process. Pop-up windows can be problematic, as explained in Floating Windows (H6), but if used judiciously, they can work well. Keep in mind that too many open windows can be annoying, and because not every link comes in the middle of a process, not all external links need to open new windows.
**SOLUTION**

External links can help build trust and credibility among your customers, while reducing the amount of work required to create new content. Take special care to ask permission from external site operators, and learn their policies on page addresses and dynamic content creation so that you can avoid most link rot. Let your customers know that they’re about to be sent to an external site by explicitly marking each link, or by putting external links in a well-marked area on your page. Use pop-up windows for external links only when the context of your site must be maintained so that customers don’t lose their place in a process.

![Figure K8.2](image)

Group external links at the end of a page (left) or in a module separated from the actual content (right).

**OTHER PATTERNS TO CONSIDER**

Use DESCRIPTIVE, LONGER LINK NAMES (K9) so that customers will know that they’re going to another site, or put all external links in their own CONTENT MODULES (D2).
User Interface Design Patterns

User Interface Design patterns are recurring solutions that solve common design problems. Design patterns are standard reference points for the experienced user interface designer. This site will help you in two ways. You can read insightful design pattern articles and browse screenshot examples.

Pattern categories

- **Getting input**
  - Getting the user to input data is a task that should be tailored to the context of use.

- **Navigation**
  - The user needs to locate specific features and content and needs navigation to accomplish this.

- **Dealing with data**
  - Data can be searched, formatted, overviewed, and browsed in a variety of ways.

- **Social**
  - Allow the user to associate, communicate, and interact with other people online.

- **Miscellaneous**
  - Patterns that haven't found their main category yet.

Recently added patterns

- **Paywall**
  - The user needs to pay to get access to a restricted area on a website.
  - **Slideshow**
  - The user needs to skim through stories without scrolling or any other unnecessary mouse movements.
  - **Gallery**
  - The user needs to browse a series of multiple images in a high resolution.

Recent blog posts

- **Designing for push and pull in web design**
  - Written by [Hubert] on Dec 4, 2011
  - The profession of improving usability asks how we best help our users fulfill their goals. The goal and task orientation of usability is carved into stone in the ISO standard definition for what usability is. There is however a part of the usability profession that can't be concerned with a clear goal.
Blank Slate

Problem summary
The user needs to get started with help or by getting a feel of how the application will feel when fully in function and filled with data.

Example

Account overview
Welcome to your motigo account. Enjoy your stay. If you need help you might find the help links listed in the right side of the page useful.

Your motigo services
- Create a new counter
- Create a new guestbook
- Create a new calendar
- Create a new forum
- Create a new shorturl

Once you have created a service, your account overview will look like this...

Forums (3)
- Support forum
- Feature discussions
- Management forum
- Create a new forum

Guestbooks (3)
- Motion guestbook

Usage
- Use when your application feels without life and empty before the user has started to enter data into the system.
- Use when you want to motivate the user to start using the system.
- Use when you want to aid the user in getting started with your web application.
- Use when you want to show best practices of using your web application.
- Use when you want to guide the users to get a good start with your web application.

GETTING INPUT

Forms
- Forming Format
- Structured Format
- Fill In The Blanks
- Input Prompt
- Good Defaults
- Captcha
- Inplace Editor
- WYSIWYG
- Live Preview
- Password Strength Meter
- Input Feedback
- Calendar Picker

Explaining the process
- Steps Left
- Inline Help Box
- Wizard
- Tour
your spring break swim destination. | buy one, get one 50% off for the family.*

women's | men's | kids' | baby

the ultimate SPRING BREAK EVENT

up to 20% off convertible car seats.*

important notice: data incident involving certain guest information.

buy one, get one 50% off on sandals & sunglasses.*

spring break must-haves.

save 20% when you spend $125 on furniture,