UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Patrones de Diseño

Agustín J. González

ElO329

Introducción

- Un patrón de diseño es una solución general reusable a un problema común recurrente.
- No es una solución definitiva directamente usable en el código, sino es una descripción sobre cómo resolver el problema.
- En electrónica existen circuitos recomendados para realizar tareas comunes como amplificar señales, crear filtros, etc. Los patrones de diseño son algo similar, pero con menor detalle.
- Es más natural observar su utilidad cuando ya hemos trabajado en varios proyectos.

Beneficios

- Usar estructuras de soluciones ya pensadas conduce a diseños más robustos en menor tiempo de desarrollo.
- Si en la documentación se señala el uso de un patrón de diseño, quienes re-visiten el proyecto a futuro saben qué cosas esperar en la solución.

Tipos de patrones de diseño

- Los patrones de diseño se han clasificado según su propósito en patrones para crear objetos, patrones para definir relaciones entre objetos (se conoce como patrón estructural), patrones de comportamiento y patrones para manejar concurrencia.
- Un paradigma o arquitectura común y previa al concepto de patrón de diseño es la arquitectura Modelo-vista-controlador usado para el desarrollo de sistemas con interfaces gráficas. Este modelo fue usado en la tarea 2 y 3.

Ejemplo: Singleton

- Veremos un patrón de diseño conocido como singleton y el patrón de arquitectura Modelo-vista-controlador
- Singleton: es usado cuando se desea crear sólo una instancia de un determinado objeto.
- Este patrón ha recibido críticas, por ejemplo, porque introduce un estado global para la aplicación.
- Este patrón se usa cuando por alguna razón deseamos tener acceso a una misma instancia cada vez.
- Lo usamos cuando crear más de una instancia sería un error.

Singleton

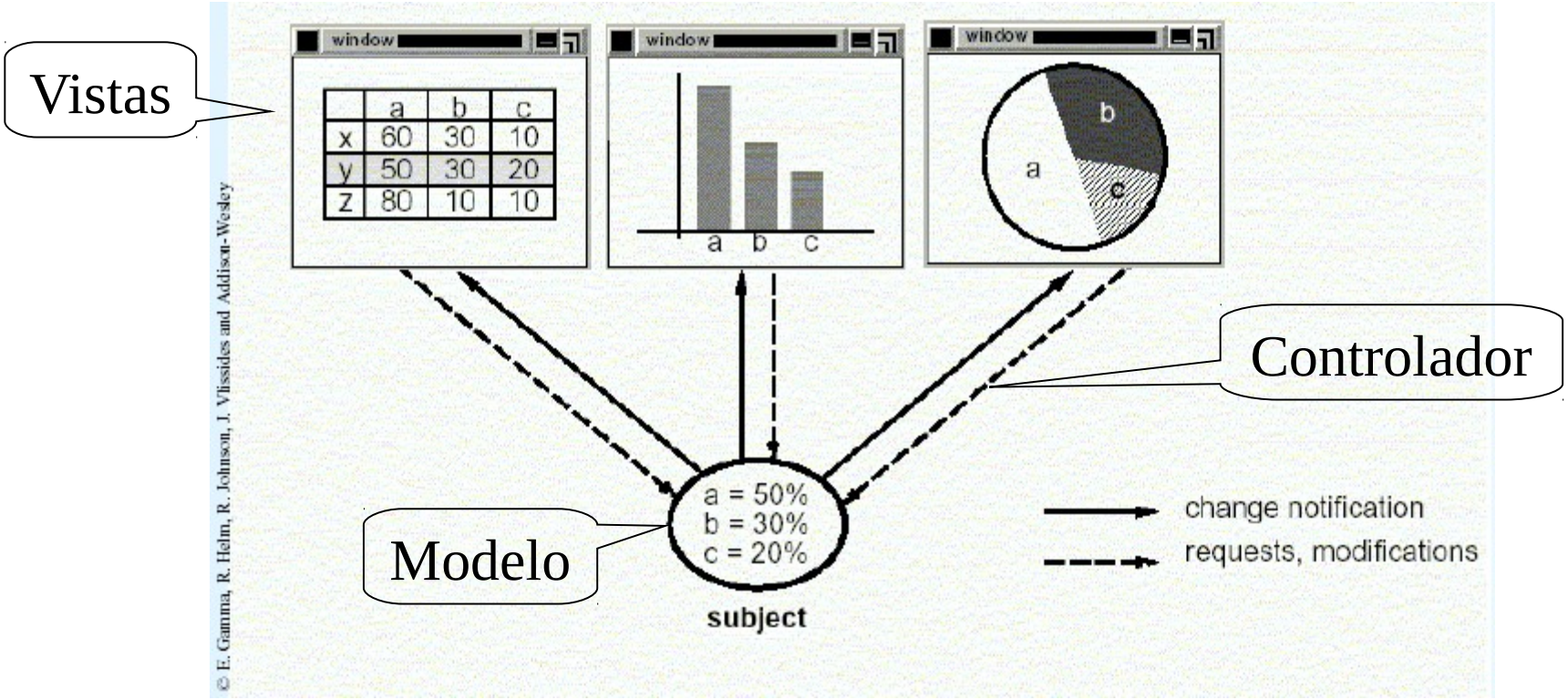
```
class Singleton {
private:
  static Singleton _instance;  // the unique instance of the class
   Singleton() {}
  ~Singleton() {}
  Singleton(const Singleton &);            // intentionally undefined
  Singleton & operator=(const Singleton &); // intentionally undefined
public:
  static Singleton &getInstance();
};
 // Source file (.cpp)
// Static member initialization.
Singleton Singleton::_instance;
Singleton &Singleton::getInstance() {
  return _instance;
}
```

Singleton

- Esta implementación del singleton posee algunos aspectos mejorables, por ejemplo: Ya sea que se cree o no instancias de esta clase, siempre existirá una ocupando espacio de memoria.
- Hay otras versiones de singleton más elaboradas que resuelven estos problemas, ver por ejemplo Wikipedia.

Modelo vista controlador

- Es un patrón estructural



Ver ejemplo en página del ramo.