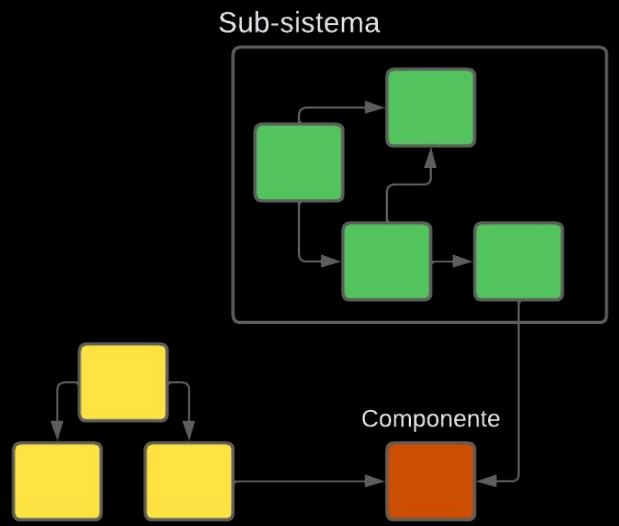
Patron de Software Facade

—

Juan Carlos Aguilar Torres
Carlos Antonio Sanchez Blanco

Definicion

Patron de Software estructural el cual busca simplificar el uso de un **sub-sistema** complejo, donde todas las operaciones complejas para usar el modulo son hechas por **un solo componente**, como un wrapper



Los patrones estructurales son enfocados a la:

-Relacion entre **componentes** (grado de cohesion y acoplamiento)

-Los **sub-sistemas** formados

Problemas

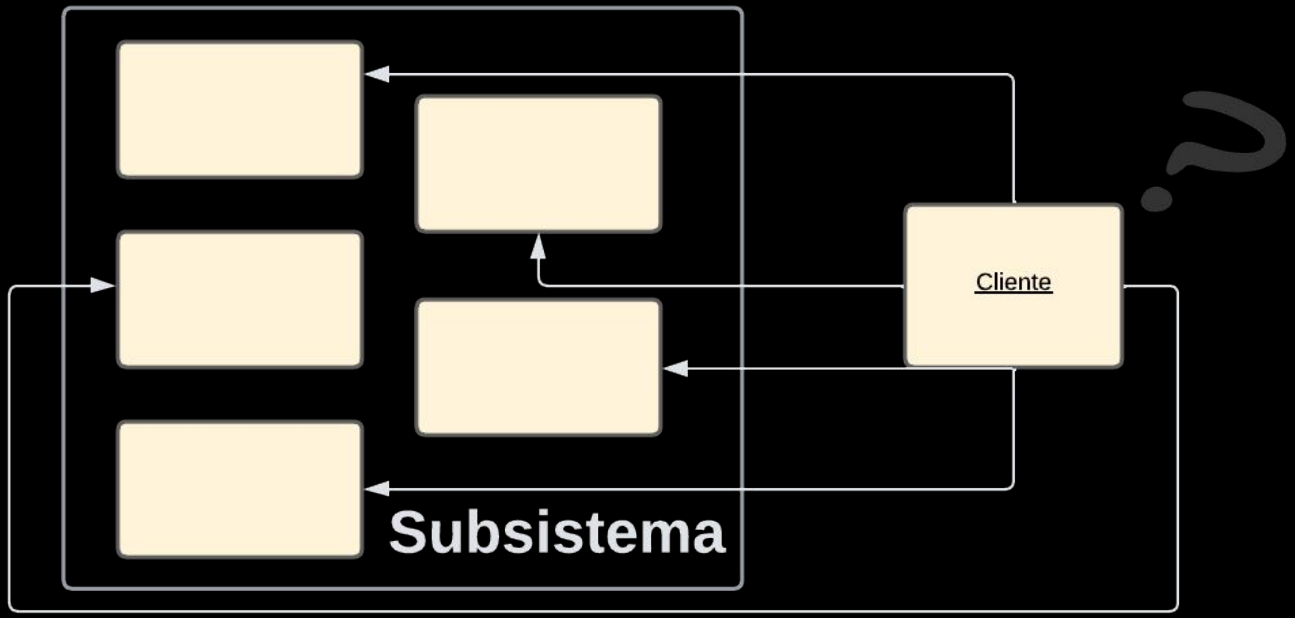
Los problemas que Facade enfrenta son:

-Sub-sistemas complejos

-Desmonolitizar un programa

-Escalado del codigo y bajar el acoplamiento

Ejemplo de problema

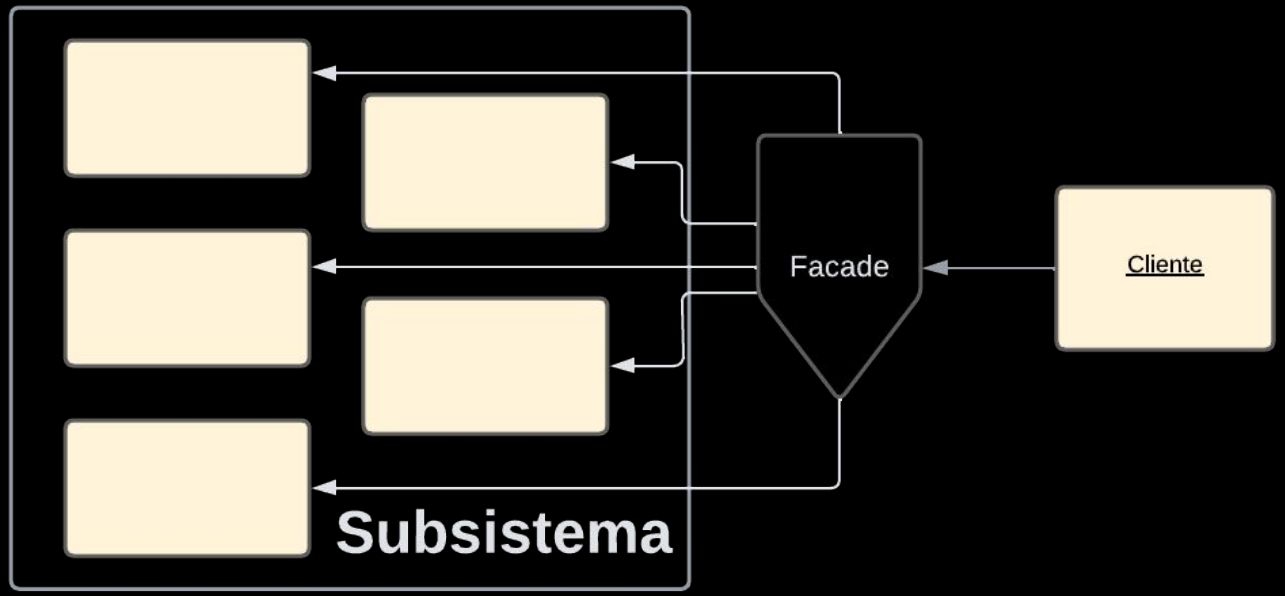


Se tiene un sub-sistema **altamente cohesivo**, **desacoplado** y funcional

Pero para poder usarlo se requiere que todos los componentes colaboren

Ahora la alta cohesion y bajo acoplamiento **causan problemas**

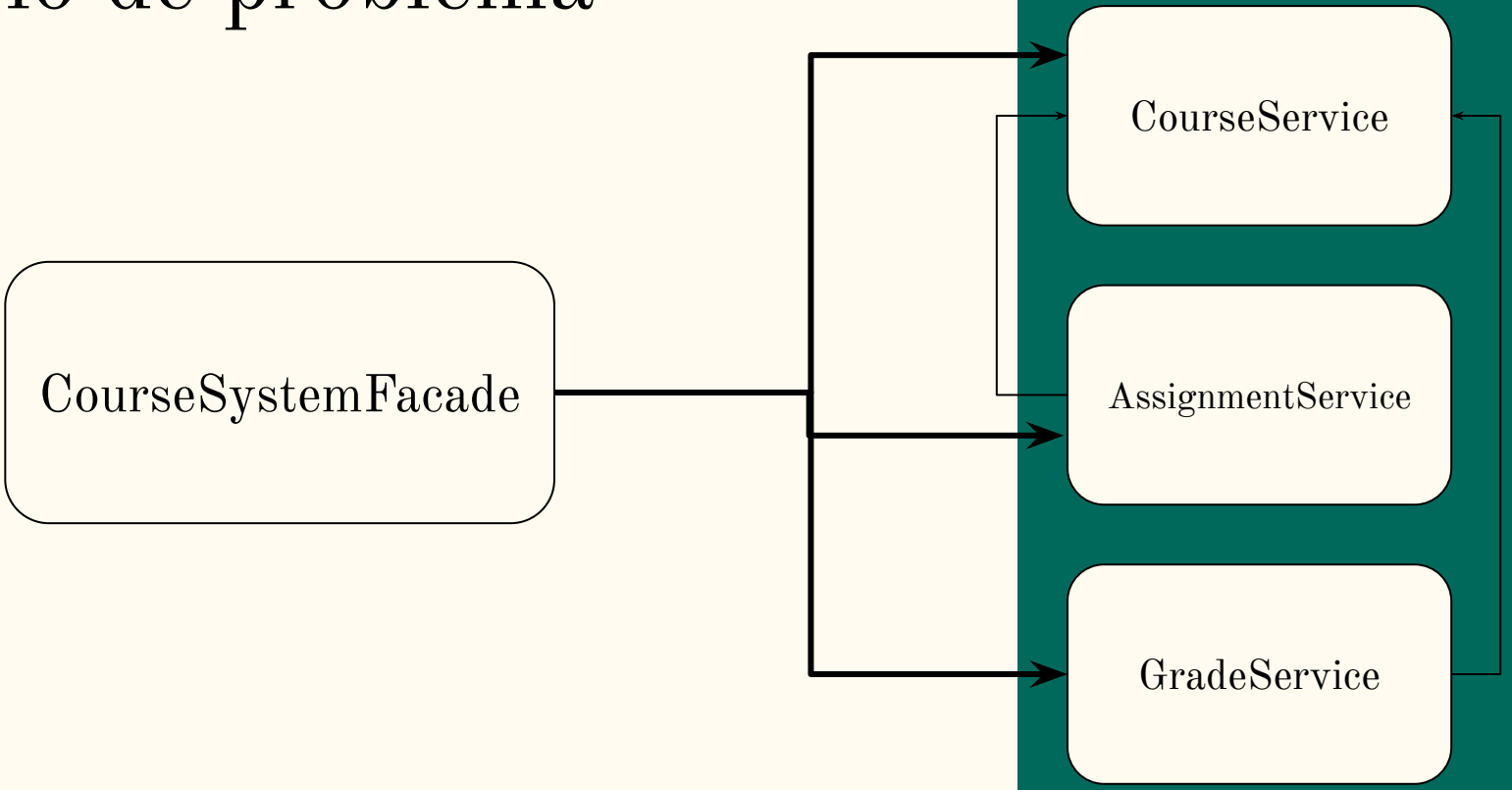
Ejemplo de problema



Antes
La implementacion la hacia el programador y escribia cada clase y sus metodos

Ahora
El Facade se encarga de la implementacion y el codigo cliente solo escribe una funcion y recibe lo mismo

Ejemplo de problema



Ejemplo en codigo

CourseSystemFacade

```
class CourseService {
    string availableCourses;

    CourseService() {
        availableCourses = "Math
                            Science
                            Literature
                            Sports";
    }
    string getAvailableCourses()
        return availableCourses;
    string enrollInCourse(string student, string courseId) {
        if (courseId in availableCourses)
            return "Student enrolled in course {courseId}";
        return "Student not enrolled";
    }
};
```

AssignmentService

GradeService

Ejemplo en codigo

CourseSystemFacade

CourseService

```
class AssignmentService {
    AssignmentService() {
        // CourseService initialization
    }

    string submitAssignment(string student, string courseId
        , string submission, CourseService& courseService) {

        if (courseId in coursesService.availableCourses) {
          return "Assignment {submission} submitted correctly";
        }
        return "Assignment not submitted";
    }
};
```

GradeService

Ejemplo en codigo

CourseSystemFacade

CourseService

AssignmentService

```
class GradeService {
    GradeService() {
        // GradeService initialization
    }

    string getGradesForCourse(string  student, string courseId
        , CourseService& courseService) {

        if (courseId in courseService.availableCourses) {
            return "Student passed :)";
        }
        return "Student did not pass ;(";
    }
};
```

Ejemplo en codigo

```
class CourseSystemFacade {
    CourseService* courseService;
    AssignmentService* assignmentService;
    GradeService* gradeService;
    string user;

    CourseSystemFacade(string user) {
        this->user = user;
        this->courseService = new CourseService();
        this->gradeService = new GradeService();
        this->assignmentService = new AssignmentService();
    }
    string getAvailableCourses()
        return this->courseService->getAvailableCourses();
    string enrollInCourse(string courseId)
        return this->courseService->enrollInCourse(this->user, courseId);
    string getGradesForCourse(string courseId)
        return this->gradeService->getGradesForCourse(user, courseId, *courseService);
    string submitAssignment(string assignmentId, string submission)
        return this->assignmentService->submitAssignment
            (this->user, assignmentId, submission, *this->courseService);
};
```

CourseService

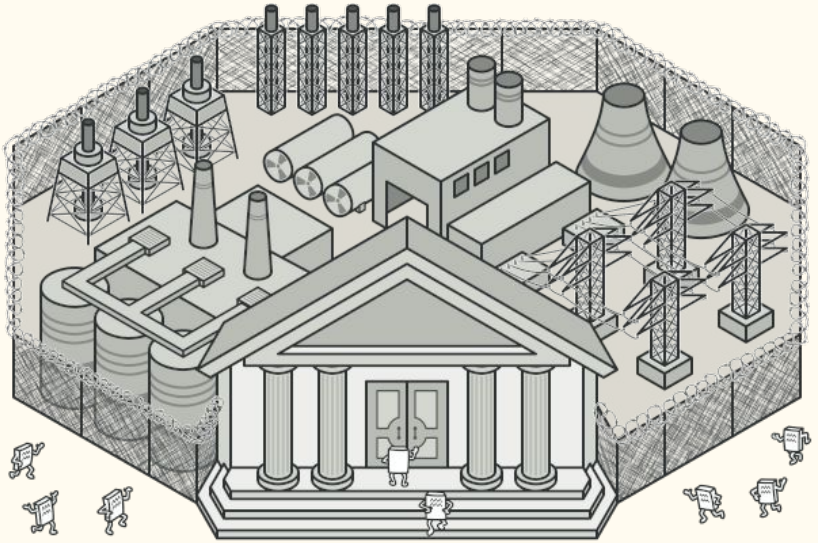
AssignmentService

GradeService

Consecuencias

El patron logra:

1. Simplificar un sub-sistema
2. Desacopla codigo
3. Escala el codigo
4. Permite las capas en un sistema



Consecuencias

Pero, lo malo es:

1. Limitacion de funciones
2. Puede dar alto acoplamiento dentro del sistema si se usa mal
3. Apertura para malas practicas
4. Puede resultar en mucha indireccion de funciones

Sugerencias

Dar cohesion en el sistema para un mejor Facade

Mantenerlo simple, evitar sobrecargar el Facade con metodos que pueden ocasionar mucho mas acoplamiento

En sistemas con varios procesos largos, pueden implementarse capas de Facades para bajar el acoplamiento dentro del sistema

El rendimiento puede verse afectado por la necesidad devarias clases de por medio para hacer algo, como en el caso para capas de Facades

Patrones relacionados

Composite

Los 2 se encargan de simplificar grupos de componentes relacionados, pero uno se **especializa en tratar a individuos y grupos como lo mismo** y el otro solo los unifica como equipo

Decorator

Los 2 se enfocan en dar funcionalidad a lo ya existente, pero uno **añade funcionalidad a una clase** mientras que el otro **encapsula clases**

Proxy

Los 2 controlan el acceso a clases, pero uno lo hace a **una sola clase** mientras que el otro **a un conjunto**

Referencias

{Disenho de software}. (Agosto 2021). Object Oriented Software Design. Recuperado el 6 de mayo de 2023
https://en.wikiversity.org/wiki/Object_Oriented_Software_Design

Refactoring Guru. (s.f.). Facade. Recuperado el 6 de mayo de 2023
https://refactoring.guru/design-patterns/facade

{Patron de software Facade}. (Enero 2023). Facade pattern. Recuperado el 6 de mayo de 2023
https://en.wikipedia.org/wiki/Facade_pattern

Actividad