

Patrones de Diseño: Facade

Análisis & Diseño de Software/Fundamentos de Ingeniería de Software



Pablo Cruz Navea-Gastón Márquez-Hernán Astudillo
Departamento de Informática
Universidad Técnica Federico Santa María

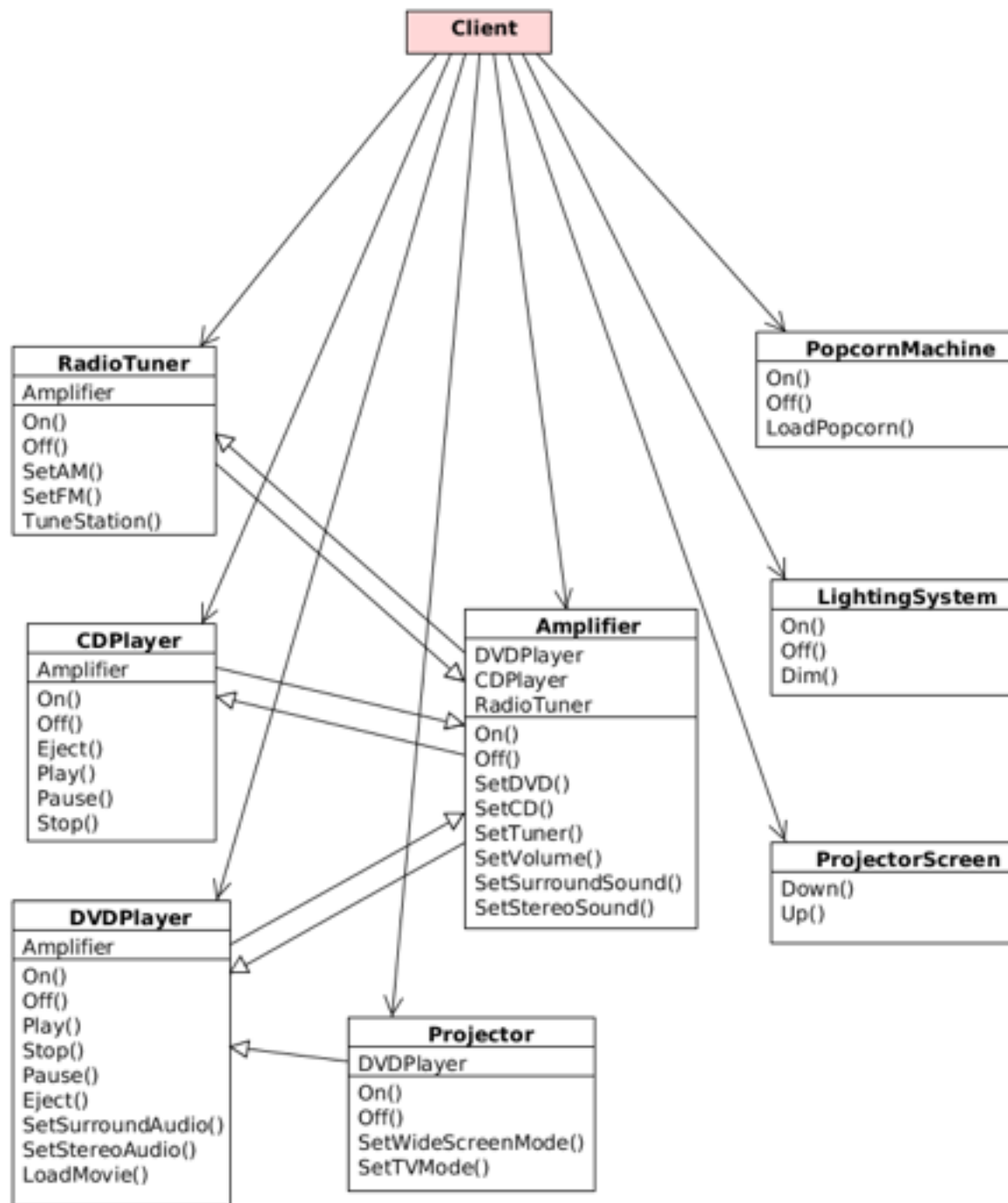
Customer service Facade

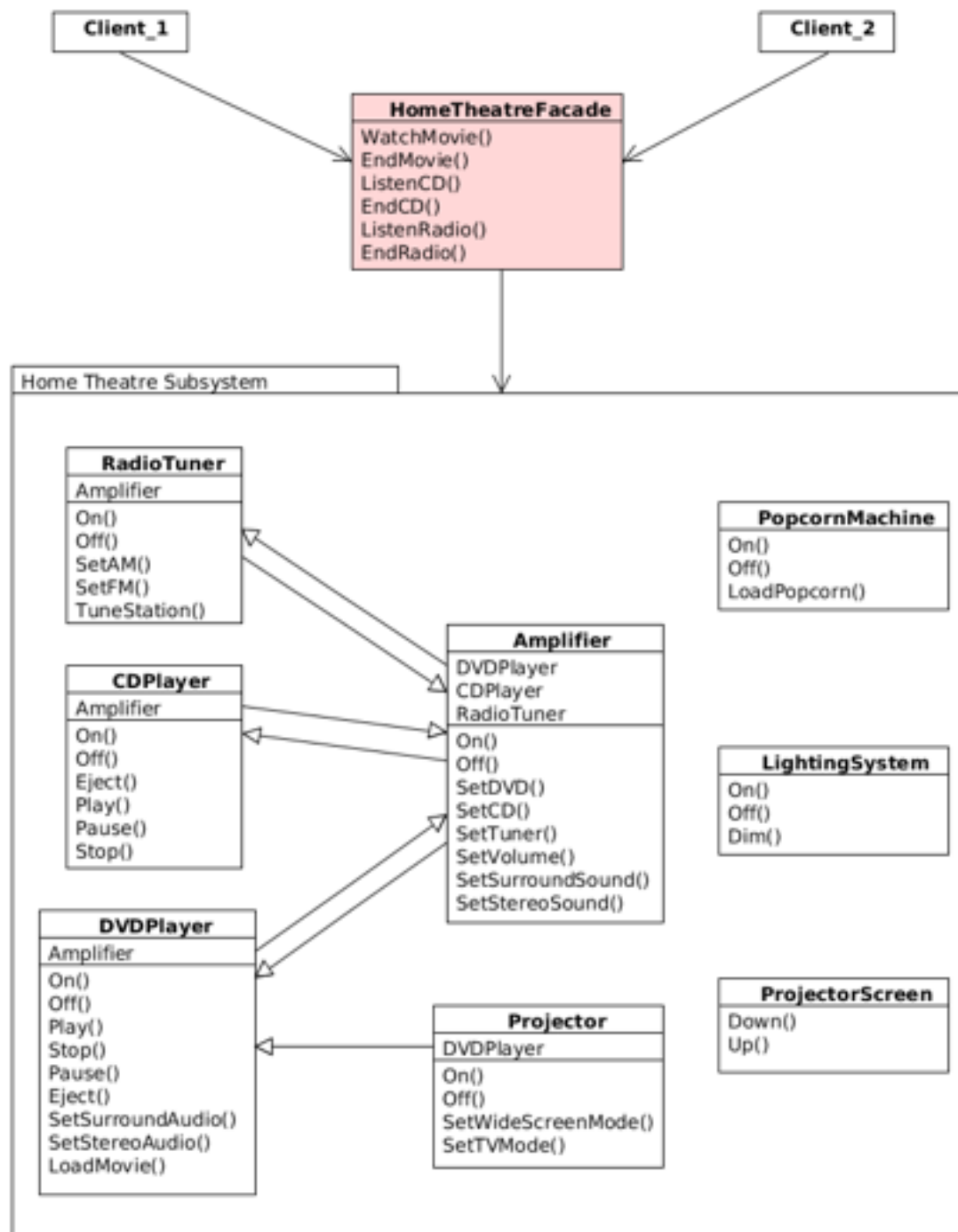


Order fulfillment

Billing

Shipping





Facade [1]

- Patrón de diseño estructural
- Propósito: proveer una interfaz única de fachada para un conjunto de otras interfaces
 - Puede entenderse como un nivel más de abstracción que facilita el uso de subsistemas



Facade [2]

- La clase Facade provee un método de fachada
- Es de fachada porque en realidad está ocultando al cliente el trabajo con un subsistema
- Por ejemplo, la clase TicketFacade puede tener un método llamado buyTicket () (método fachada) que en realidad hace llamados a otros métodos:
 - checkAvailability () en clase Ticket
 - assignPassenger () en clase Ticket
 - registerPayment () en clase Payment

Ejemplo [1]

```
public class Class1 {
```

```
    public int doSomethingComplicated(int x) {  
        return x * x * x;  
    }
```

```
}
```

```
public class Class2 {
```

```
    public int doAnotherThing(Class1 class1, int x) {  
        return 2 * class1.doSomethingComplicated(x);  
    }
```

```
}
```

```
public class Class3 {
```

```
    public int doMoreStuff(Class1 class1, Class2 class2, int x) {  
        return class1.doSomethingComplicated(x) * class2.doAnotherThing(class1, x);  
    }
```

```
}
```

Ejemplo [2]

- Para un cliente que no conoce el contenido de Class1, Class2 y Class3 es complicado entender cómo interactúan estas clases
- Por lo tanto, la misión es simplificar la interacciones en el sistema con el objetivo de que el cliente pueda usar el sistema de forma simple y estandarizada.
- Facade es la solución

Ejemplo [3]

```
public class Facade {
```

```
    public int cubeX(int x) {  
        Class1 class1 = new Class1();  
        return class1.doSomethingComplicated(x);  
    }
```

```
    public int cubeXTimes2(int x) {  
        Class1 class1 = new Class1();  
        Class2 class2 = new Class2();  
        return class2.doAnotherThing(class1, x);  
    }
```

```
    public int xToSixthPowerTimes2(int x) {  
        Class1 class1 = new Class1();  
        Class2 class2 = new Class2();  
        Class3 class3 = new Class3();  
        return class3.doMoreStuff(class1, class2, x);  
    }
```

```
}
```

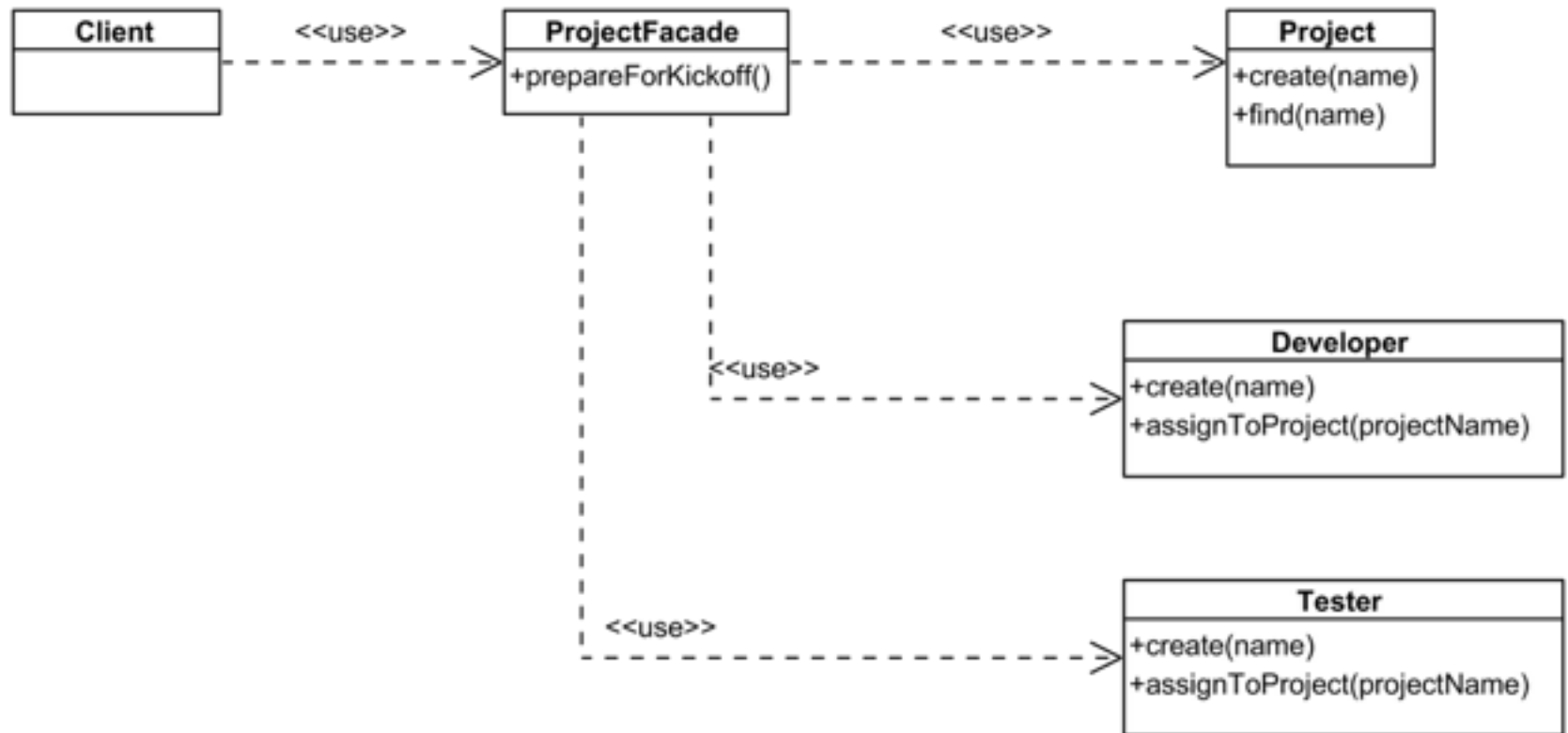
Ejemplo [4]

```
public class FacadeDemo {  
  
    public static void main(String[] args) {  
  
        Facade facade = new Facade();  
  
        int x = 3;  
        System.out.println("Cube of " + x + ":" + facade.cubeX(3));  
        System.out.println("Cube of " + x + " times 2:" + facade.cubeXTimes2(3));  
        System.out.println(x + " to sixth power times 2:" +  
facade.xToSixthPowerTimes2(3));  
  
    }  
  
}
```

Ejemplo: Facade con Java [1]

- El inicio de un proyecto (*kickoff*) requiere instrumentar el ambiente de desarrollo
- En el ejemplo veremos un caso muy sencillo de instrumentación de un Issue Tracker
- `prepareForKickoff()` es el método fachada que “oculta” al cliente:
 - Creación de un proyecto
 - Creación de un desarrollador y asignación al proyecto
 - Creación de un Tester y asignación al proyecto

Ejemplo: Facade con Java [2]



Ejemplo: Facade con Java [3]

```
public class ProjectFacade() {  
    public void prepareForKickoff() {  
        Project newProject = new Project();  
        newProject.create("FISW");  
  
        Developer newDeveloper = new Developer();  
        newDeveloper.create("Juan Pérez");  
  
        Tester newTester = new Tester();  
        newTester.create("John Doe");  
  
        newDeveloper.assignToProject("FISW");  
        newTester.assignToProject("FISW");  
    }  
}
```

Ejemplo: Facade con Java [4]

- Algunas observaciones:
 - Podríamos haber usado los constructores, pero para el ejemplo es más transparente un método especial “create”
 - Las clases Project, Developer y Tester tienen implementaciones particulares que no afectan al cliente
 - Al cliente sólo le importa la fachada!

Oracle Facade

```
import java.sql.Connection;

public class OracleHelper {

    public static Connection getOracleDBConnection(){
        //get Oracle DB connection using connection parameters
        return null;
    }

    public void generateOraclePDFReport(String tableName, Connection con){
        //get data from table and generate pdf report
    }

    public void generateOracleHTMLReport(String tableName, Connection con){
        //get data from table and generate pdf report
    }

}
```

Duda

- ¿Se puede relacionar el patrón Facade con Abstract Factory y crear un nuevo patrón, por ejemplo, *Abstract Facade Factory*?

Duda

- Facade se utiliza cuando se desea esconder la implementación.
- Abstract Factory se utiliza cuando se desea esconder detalles en la construcción de instancias

FIN