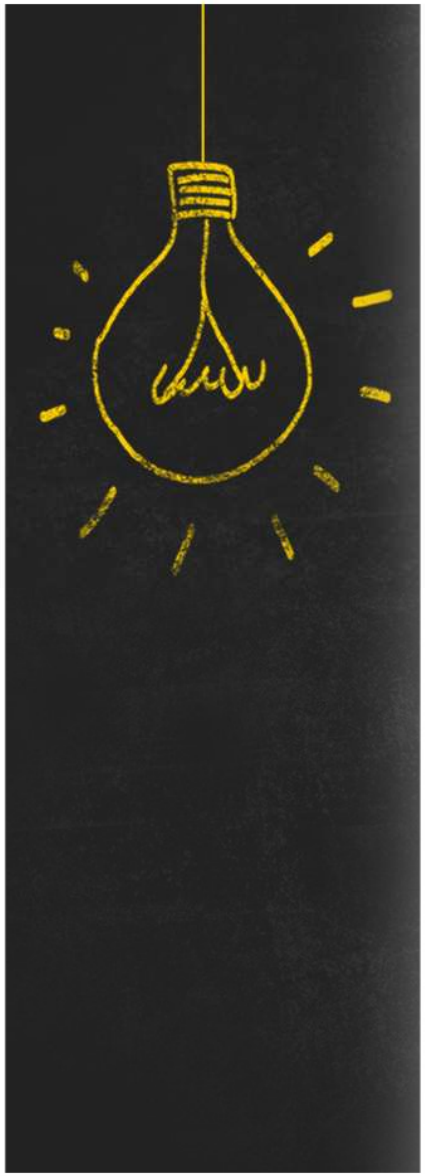
Programación 4
Guía Semana 10 (27/05)

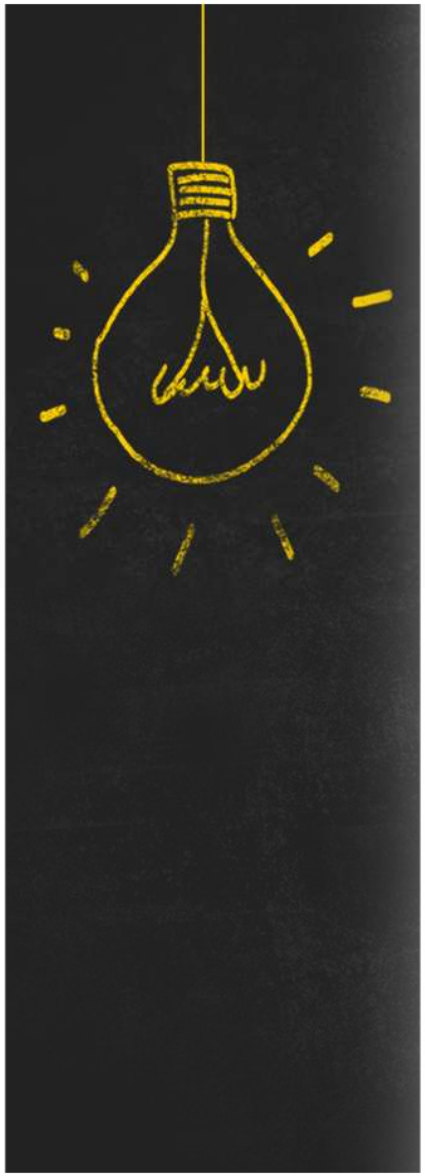
InCo, FING, UdelaR



Objetivo

Los objetivos de esta semana son:

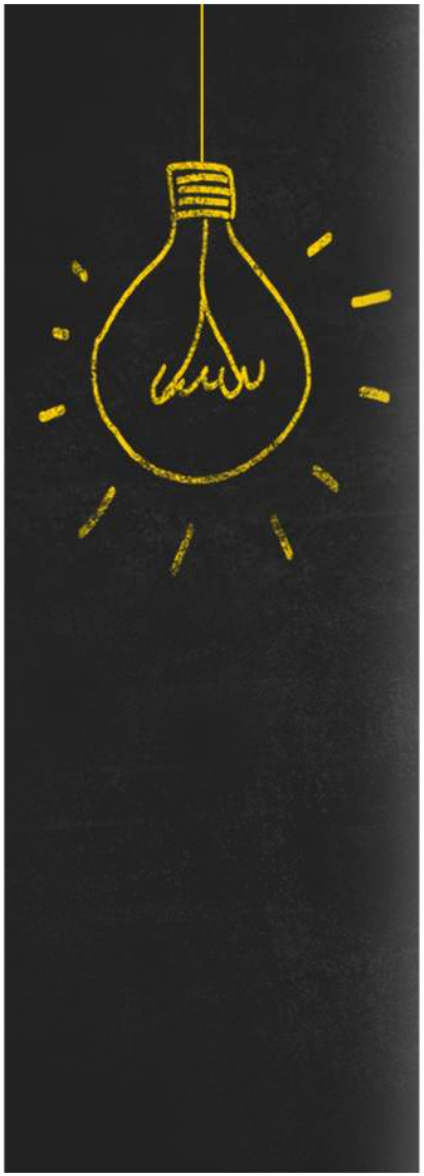
1. introducir **guías para el abordaje del diseño** de bajo nivel de forma sistemática;
2. introducir el **Diagrama de Clases de Diseño** (**DCD**) como herramienta para el diseño de estructura;
3. introducir el concepto de **Patrón de Diseño**;
4. presentar patrones de diseño básicos: **Factory**, **Singleton**, **Composite**, **State** y **Observer**.



Resumen :: Abordaje del diseño

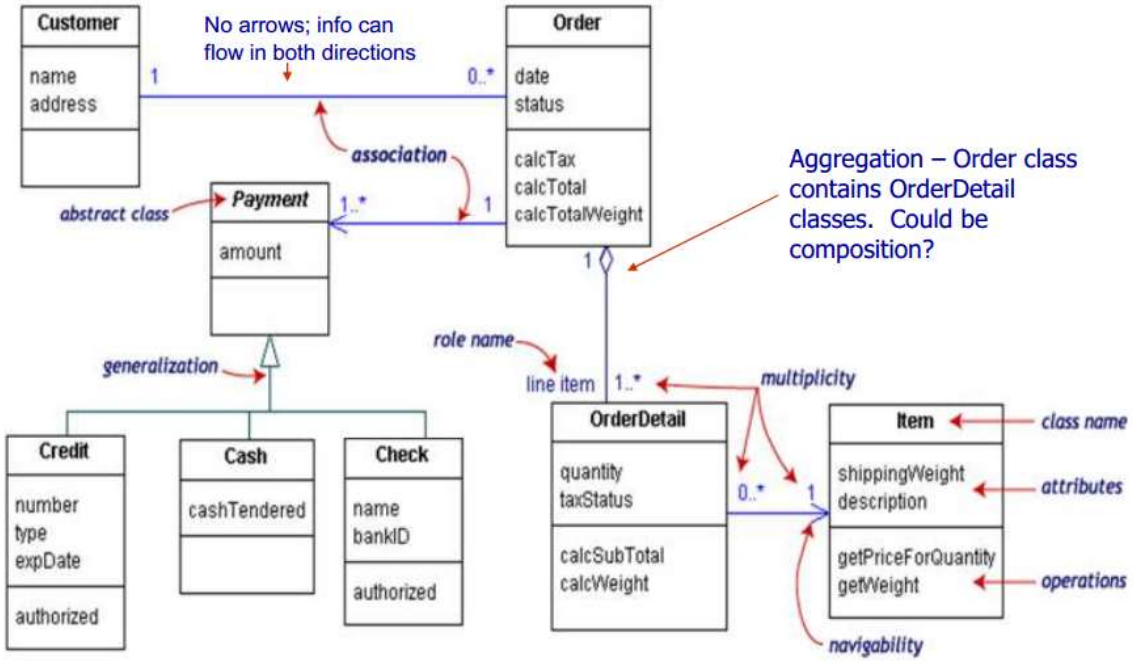
El abordaje de la etapa de diseño puede realizarse sistemáticamente

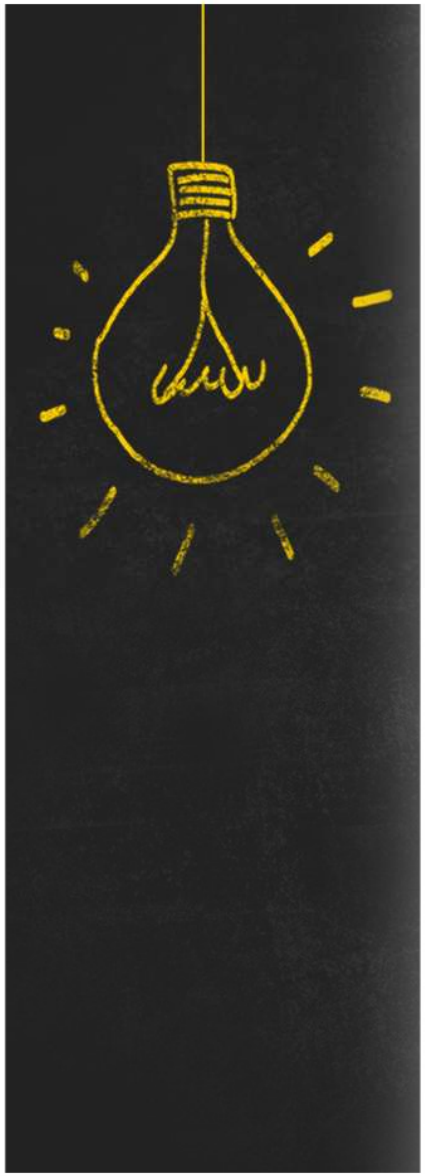
1. **Organizar Operaciones**: definir controladores, interfaces del sistema y la fábrica; organizar las operaciones según afinidad temática, afinidad funcional o casos de uso
2. **Definir Ubicación de Instancias**
3. **Definir Colaboraciones**
4. **Diseñar Colaboraciones**



Resumen :: Diag. de Clases

Un **Diagrama de Clases de Diseño (DCD)** especifica la estructura de una colaboración

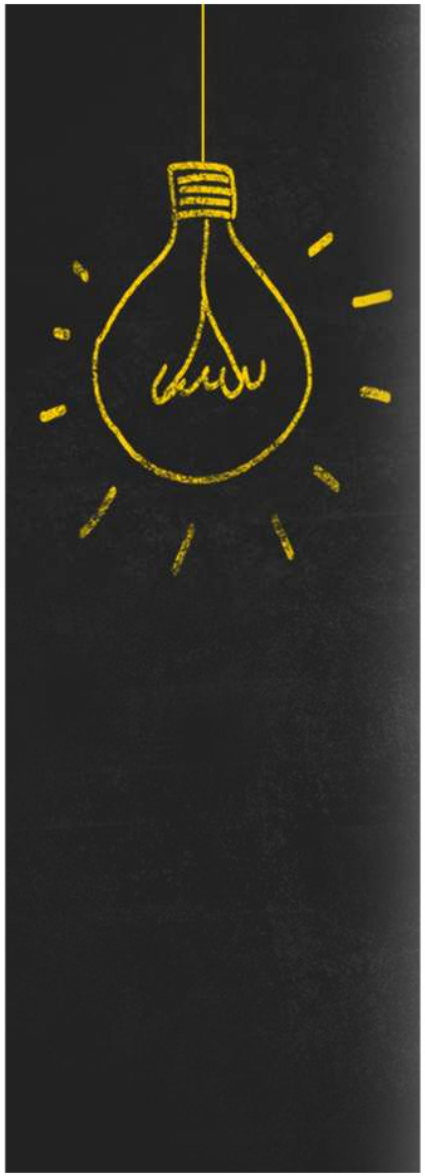




Resumen :: Patrones de Diseño

Problemas de diseño aparecen recurrentemente en diferentes proyectos, algunos ejemplos:

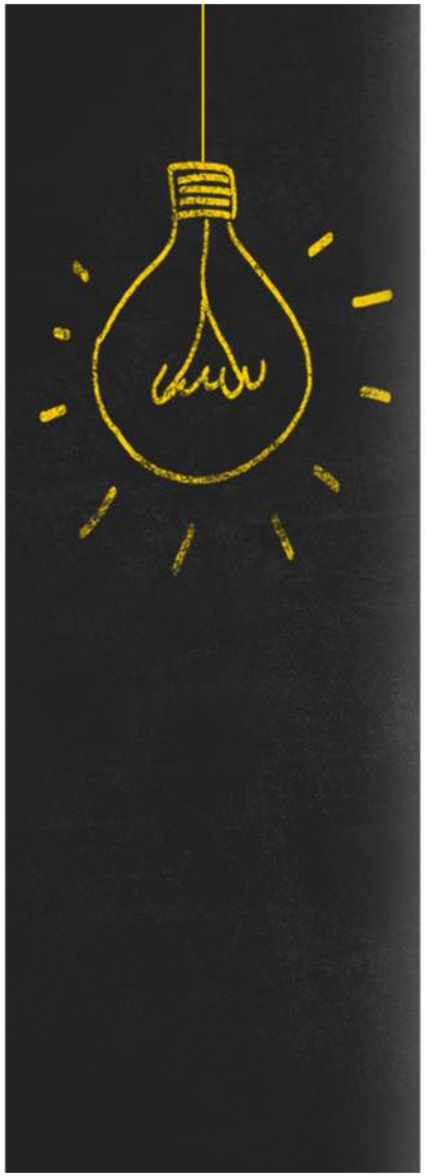
- ¿Cómo acceder a objetos sin acoplarse directamente a ellos? (**Factory**)
- ¿Cómo acceder a un conjunto de interfaces por medio de un punto de acceso común? (**Facade**)
- ¿Cómo restringir que una clase tenga una sola instancia y que se tenga visibilidad global hacia ella? (**Singleton**)
- ¿Cómo puedo manejar diferentes estados de comportamiento en un objeto? (**State**)
- ¿Cómo definir una dependencia entre objetos, de forma que cuando uno cambie de estado todos los dependientes sean notificados? (**Observer**)



Resumen :: Patrones de Diseño

Un **Patrón de Diseño** da un nombre, motiva y explica el diseño general que se aplica a un problema de diseño recurrente:

- **Problema:** explica el problema tipo y su contexto
- **Estructura:** diagrama de clases que ilustra la estructura de la colaboración abstracta que soluciona al problema tipo
- **Participantes:** descripción de las clases que forman parte de la estructura y sus responsabilidades
- **Interacciones:** diagramas de interacción que ilustran el funcionamiento de la colaboración abstracta
- **Consecuencias:** comentarios, discusiones, sugerencias y advertencias para entender e implementar el patrón

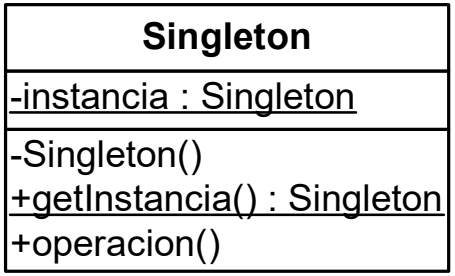


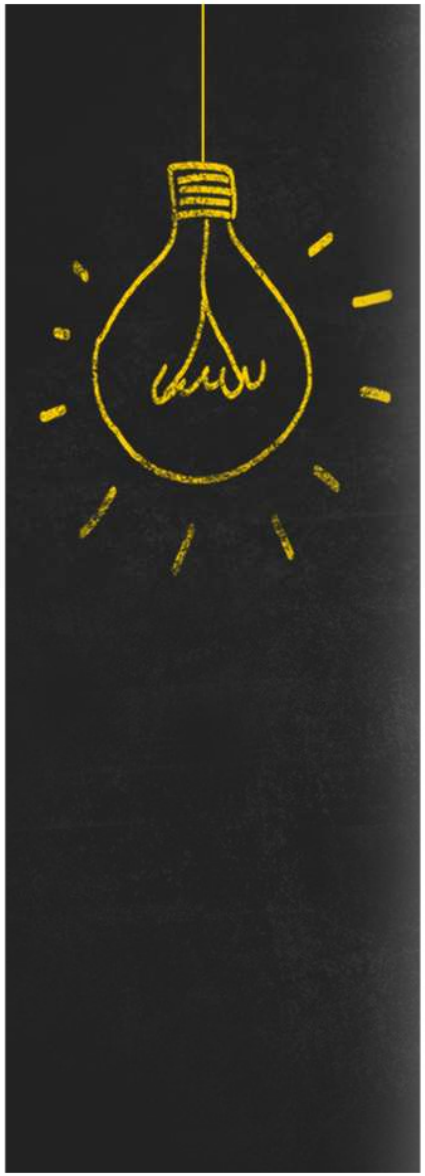
Resumen :: Singleton

- **Problema**

  “Asegurar que una clase tenga una sola instancia y proveer un acceso global a ella”

- **Estructura**



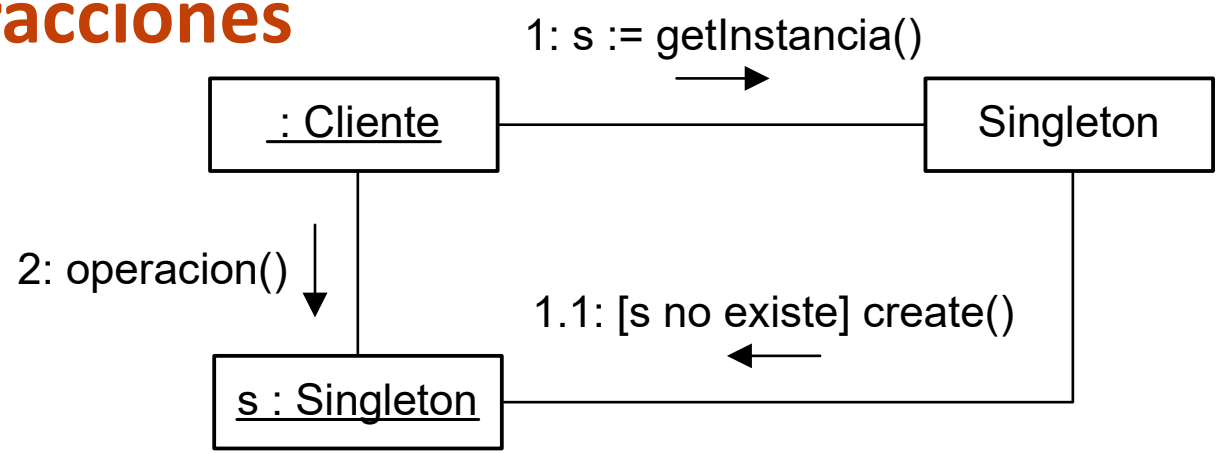


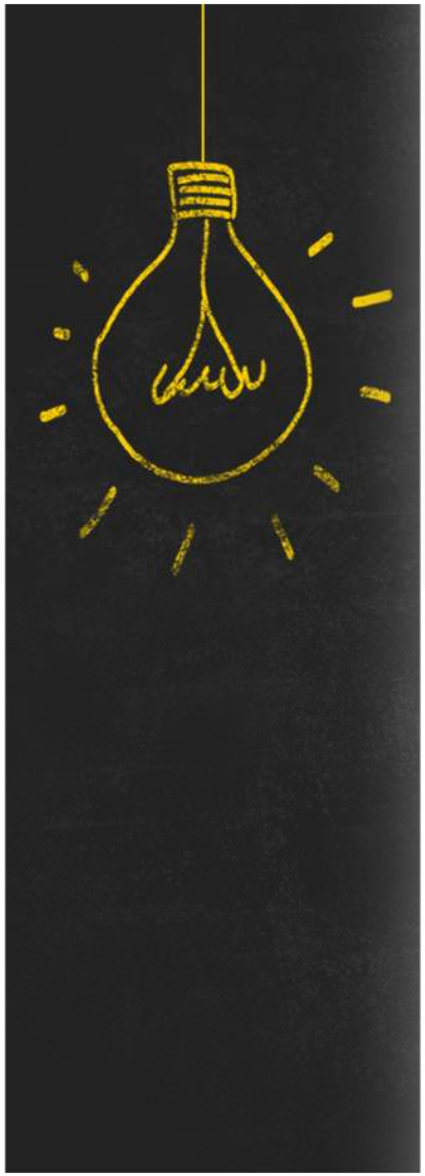
Resumen :: Singleton

- **Participantes**

  `Singleton`: provee una operación de clase (`getInstancia()`) que permite acceder a la única instancia

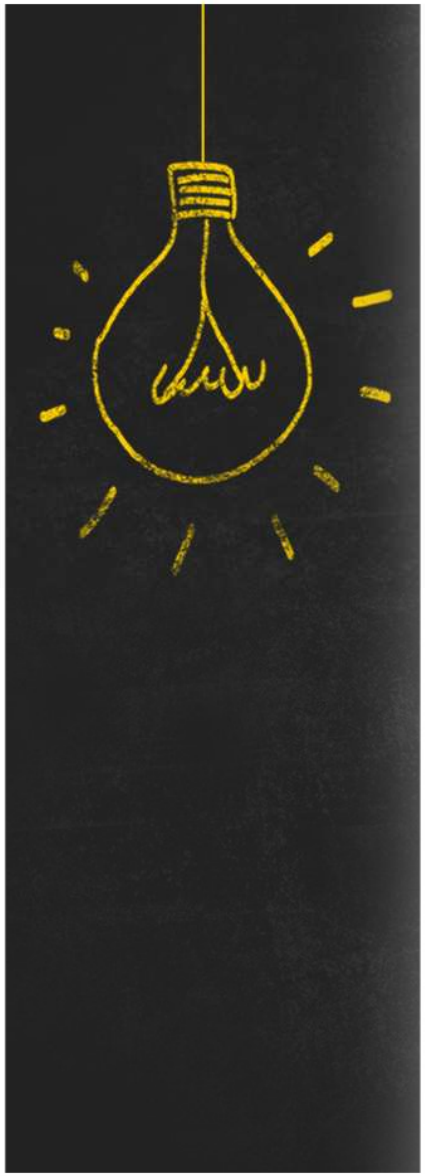
- **Interacciones**





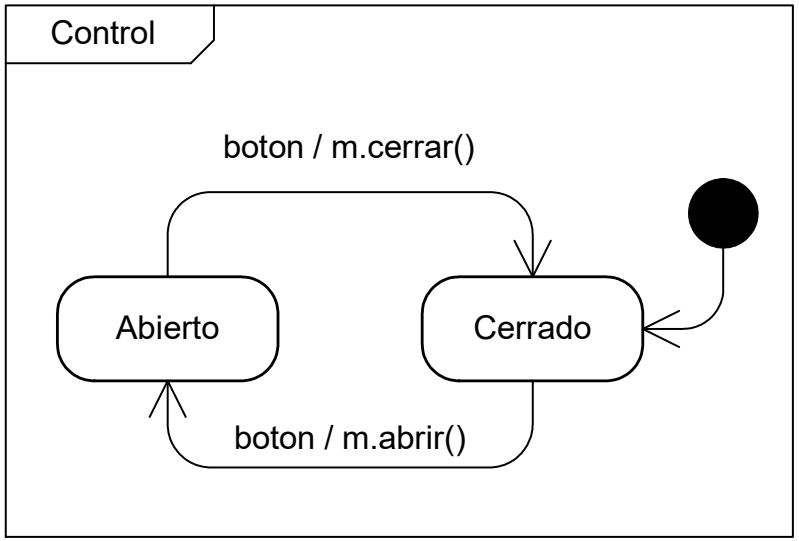
Resumen :: Singleton

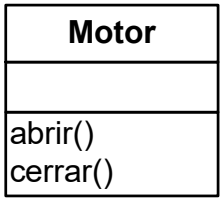
- **Consecuencias**
  - Se provee acceso controlado a una única instancia
  - Se permiten variantes en las que se varíe el número máximo de instancias
  - Un cliente puede liberar la memoria de la instancia
  - Derivar una clase Singleton resulta complejo
- **Aplicaciones**
  - Las fábricas y manejadores suelen ser Singleton
  - Algunos controladores también



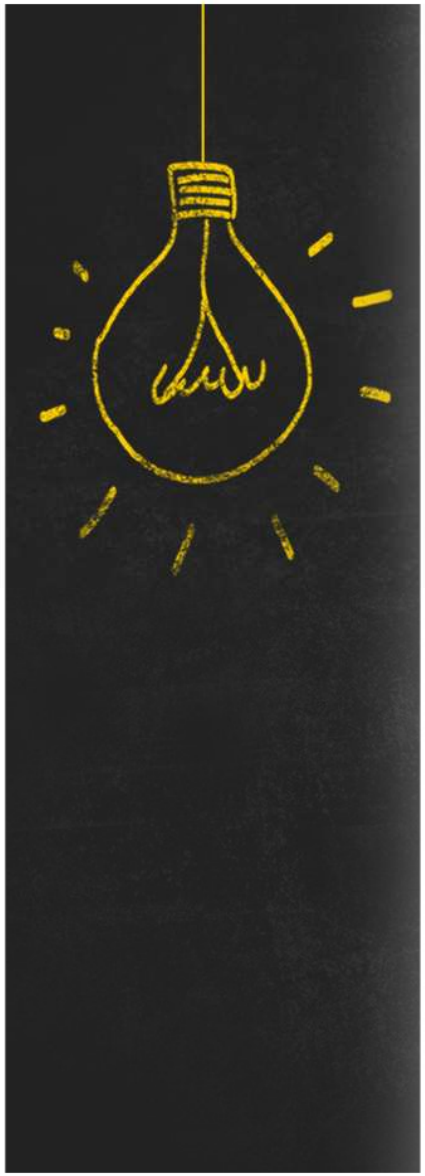
Resumen :: State

Motivación: puerta automática controlada por control remoto de un solo botón





`Motor` maneja el motor que abre y cierra la puerta

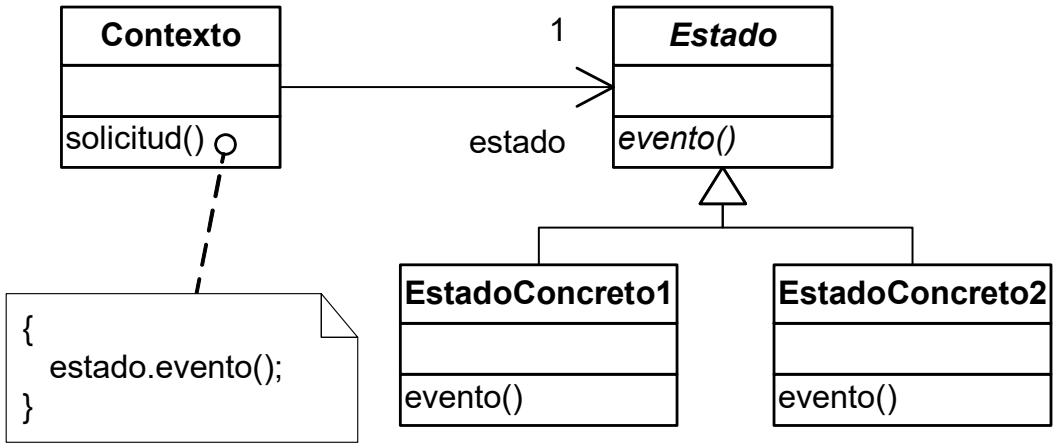


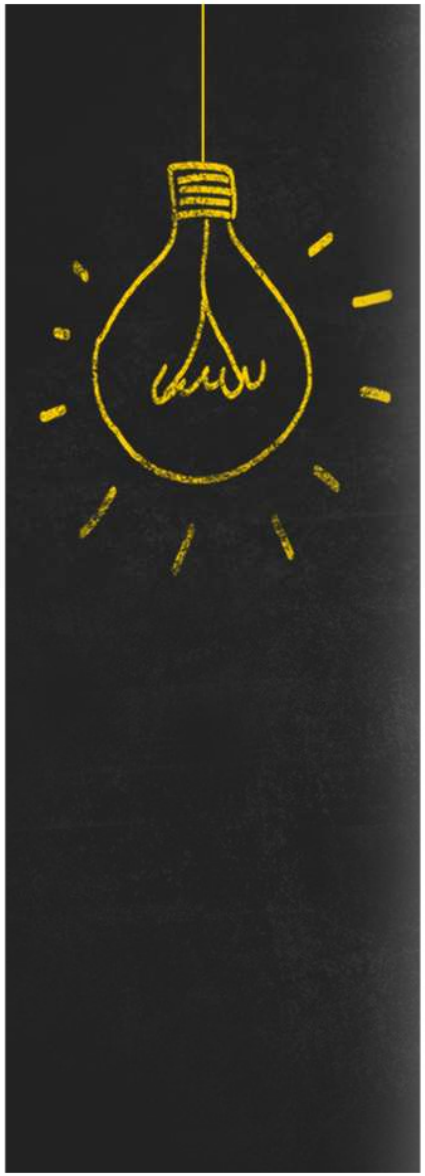
Resumen :: State

- **Problema**

  “Permitir que un objeto varíe su comportamiento cuando su estado interno cambie. El objeto parecerá haber cambiado de clase”

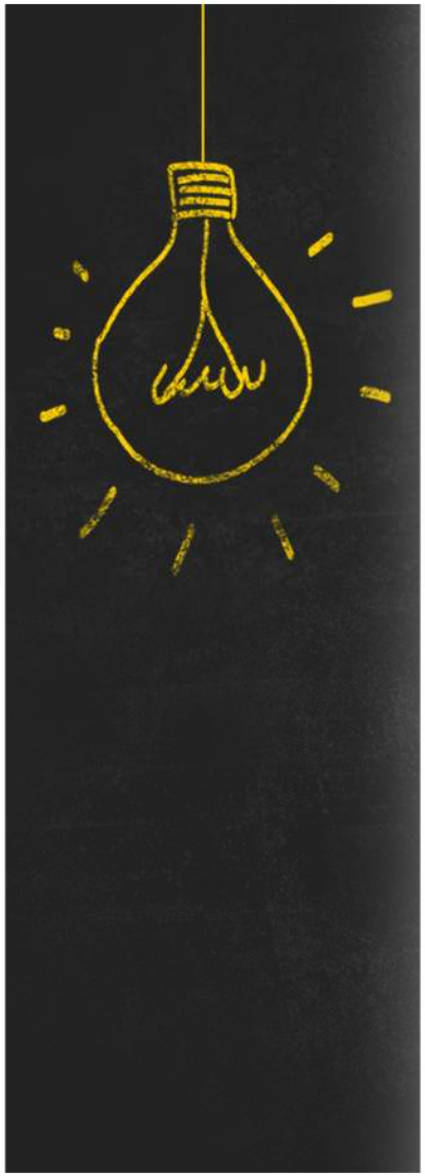
- **Estructura**





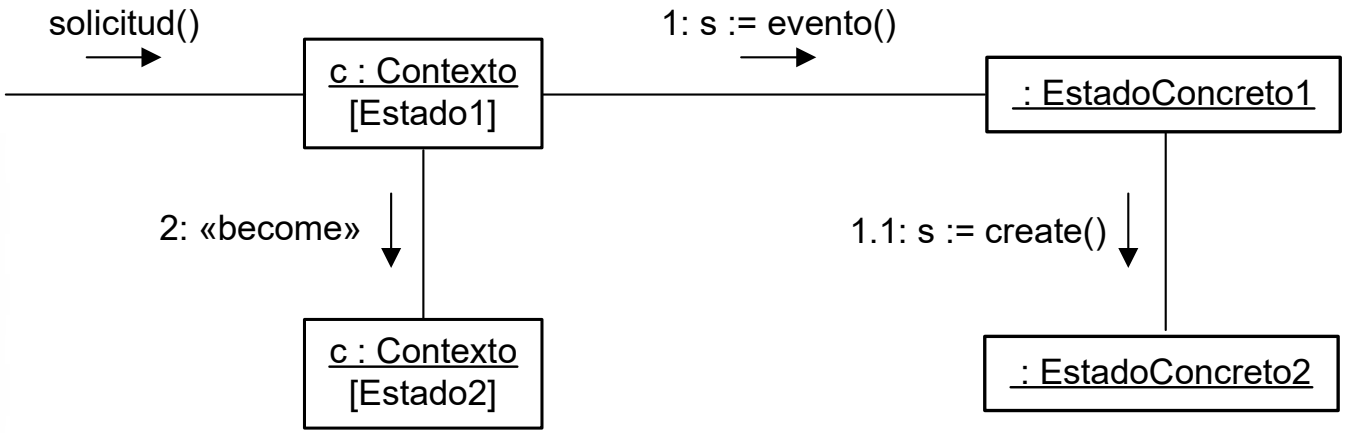
Resumen :: State

- **Participantes**
  - Contexto: es la clase de objetos cuyo comportamiento varía al cambiar el estado interno. Mantiene una referencia a un estado concreto. Delega el comportamiento variable al estado actual.
  - Estado: generaliza los diferentes estados concretos.
  - EstadoConcreto: cada una de estas clases implementa un comportamiento particular del Contexto que sea dependiente del estado.

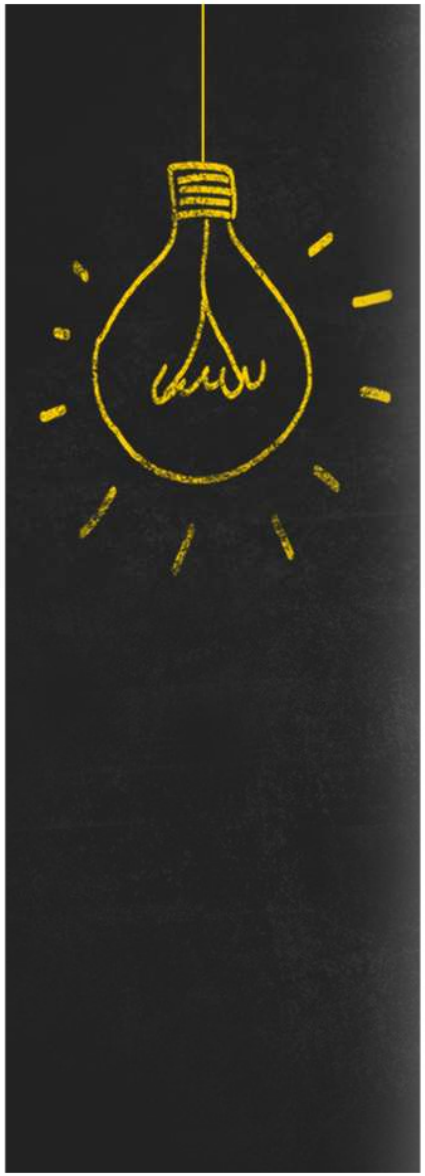


Resumen :: State

- **Interacciones**

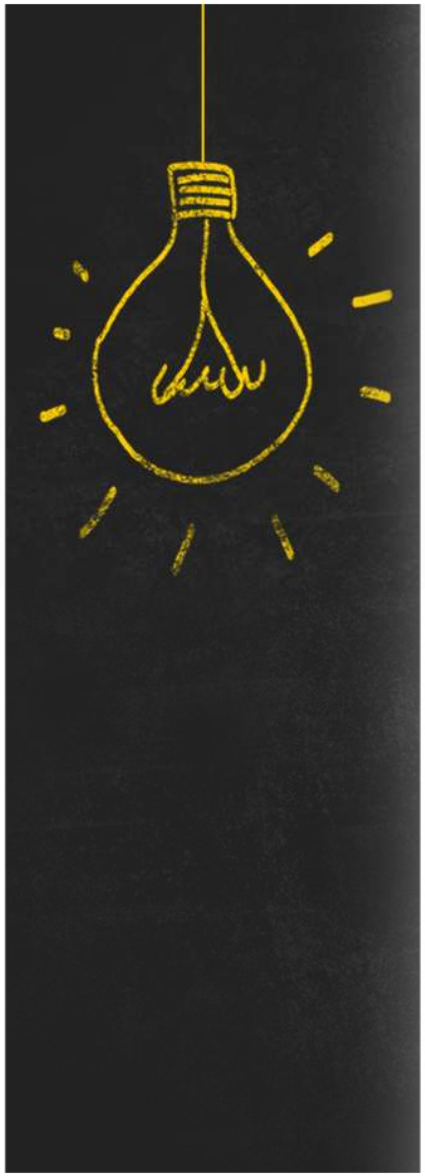


El Contexto, estando en Estado1, al recibir una “solicitud” cambia a Estado2



Resumen :: State

- **Consecuencias** (algunas)
  - Alguien debe tener la responsabilidad de eliminar, luego de una transición, la instancia que representa el estado anterior.
  - Los estados concretos pueden tener estado propio. Si no lo tienen pueden ser diseñados como Singleton.
  - En casos en que el Contexto tiene muchos estados, la cantidad de clases (a causa de los estados concretos) puede ser muy grande.
  - Es simple agregar nuevos estados, pero es necesario modificar estados concretos existentes para incluir transiciones al estado nuevo.
  - El Contexto puede pasarse como parámetro en los eventos.
  - Permite eliminar la lógica condicional del Contexto.



¿Qué hago esta semana?

1. Estudio los materiales de Teórico y las lecturas recomendadas. Las clases correspondientes se encuentran en OpenFing.
    - 13 - Diseño: Guias
    - 14 - Diseño: Estructura
    - 15 - Diseño: Patrones de Diseño
3. Realizo el Práctico 5 “Diseño, Diagramas de Clases de Diseño, Patrones de Diseño”. Están publicadas las notas de resolución de los Ejercicios 2, 7, 9 y 10. Aquí se introducen algunos patrones que no están en el teórico.
4. Finalizo el Laboratorio 3 “Diseño”.
   Plazo de entrega: lunes 03/06, 15hs