Patrones GRASP

Patrones GRASP

- Acrónimo de *General Responsibility Assignment Software Patterns.*

- Describen los principios fundamentales para asignar responsabilidades a los objetos.

El patrón “Experto” (I)

- Problema: *¿en qué debemos basarnos para asignar responsabilidades a las clases?*

- Solución: *asignar la responsabilidad al “experto en la información”*

  El *experto en la información* es la clase que tiene la información necesaria para cumplir la responsabilidad.

El patrón “Experto” (y II)

- Es un poco de perogrullo: expresa que los objetos deben hacer las cosas relacionadas con la información que poseen.

El patrón “Creador”

- Problema: *¿quién es el responsable de crear una nueva instancia de una clase?*
- Solución: *B es responsable de crear una instancia de A si:*
  - B agrega objetos de A
  - B contiene referencias a objetos de A
  - B almacena instancias de A
  - B utiliza estrechamente objetos de A
  - B tiene la información de inicialización que se necesita para crear un objeto de clase A

El patrón “Bajo acoplamiento” (I)

- Problema: *¿cómo mantener un bajo acoplamiento para lograr, entre otras cosas, alta reutilización?*

Nota: el *acoplamiento* mide el grado en que una clase está conectada a otra, tiene conocimiento de otra o, de alguna manera, depende de otra.

El patrón “Bajo acoplamiento” (II)

- Situaciones de acoplamiento:
  - X tiene un miembro o declara una variable de clase Y
  - X tiene un método que toma como parámetro un objeto de clase Y
  - X es un descendiente de Y

El patrón “Bajo acoplamiento” (III)

- Desventajas del acoplamiento:
  - Los cambios en una clase pueden implicar cambios en las clases relacionadas.
  - Dificultad de comprensión.
  - Dificultad de reutilización.
  - ¿Pérdida de rendimiento?

El patrón “Bajo acoplamiento” (IV)

- Solución: *asignar la responsabilidad de manera que el acoplamiento permanezca bajo.*

El patrón “Bajo acoplamiento” (y V)

- Consideraciones:
  - El bajo acoplamiento permite crear clases más independientes, más reutilizables, lo que implica mayor productividad
  - El acoplamiento puede no ser importante si la reutilización no está en nuestros objetivos
  - La especialización es una forma *fuerte* de acoplar clases
  - El acoplamiento bajísimo produce diseños muy pobres, objetos sobrecargados, complejos, etc.

El patrón “Alta cohesión” (I)

- Problema: *¿cómo mantener la complejidad de una clase en niveles manejables?*

Nota: la *cohesión* mide el grado en que están relacionadas las responsabilidades de una clase.

El patrón “Alta cohesión” (II)

- Desventajas de una clase con baja cohesión:
  - Difícil de comprender
  - Difícil de reutilizar
  - Difícil de mantener
  - Delicada, se afecta mucho por los cambios

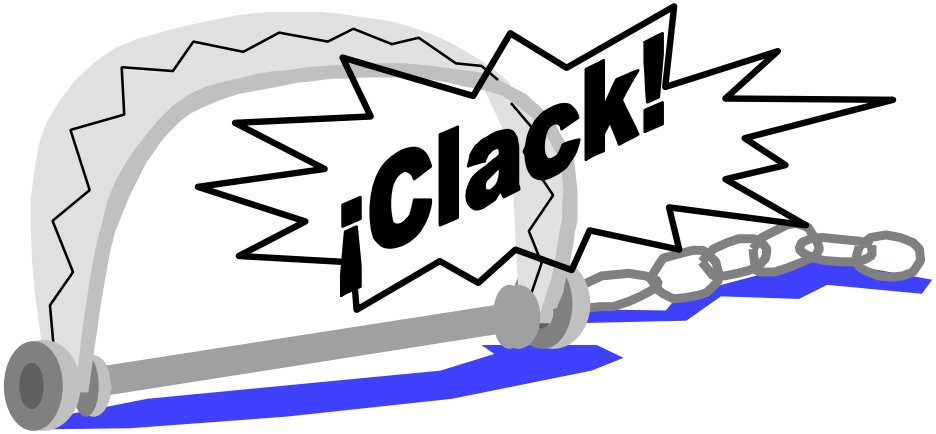
El patrón “Alta cohesión” (y III)

- Solución: *asignar responsabilidades de manera que la cohesión se mantenga alta.*

El patrón “Controlador” (I)

- Problema: *¿quién debería ser responsable de manejar un evento del sistema?*

Nota: Un *evento del sistema* es un evento generado por un actor externo.



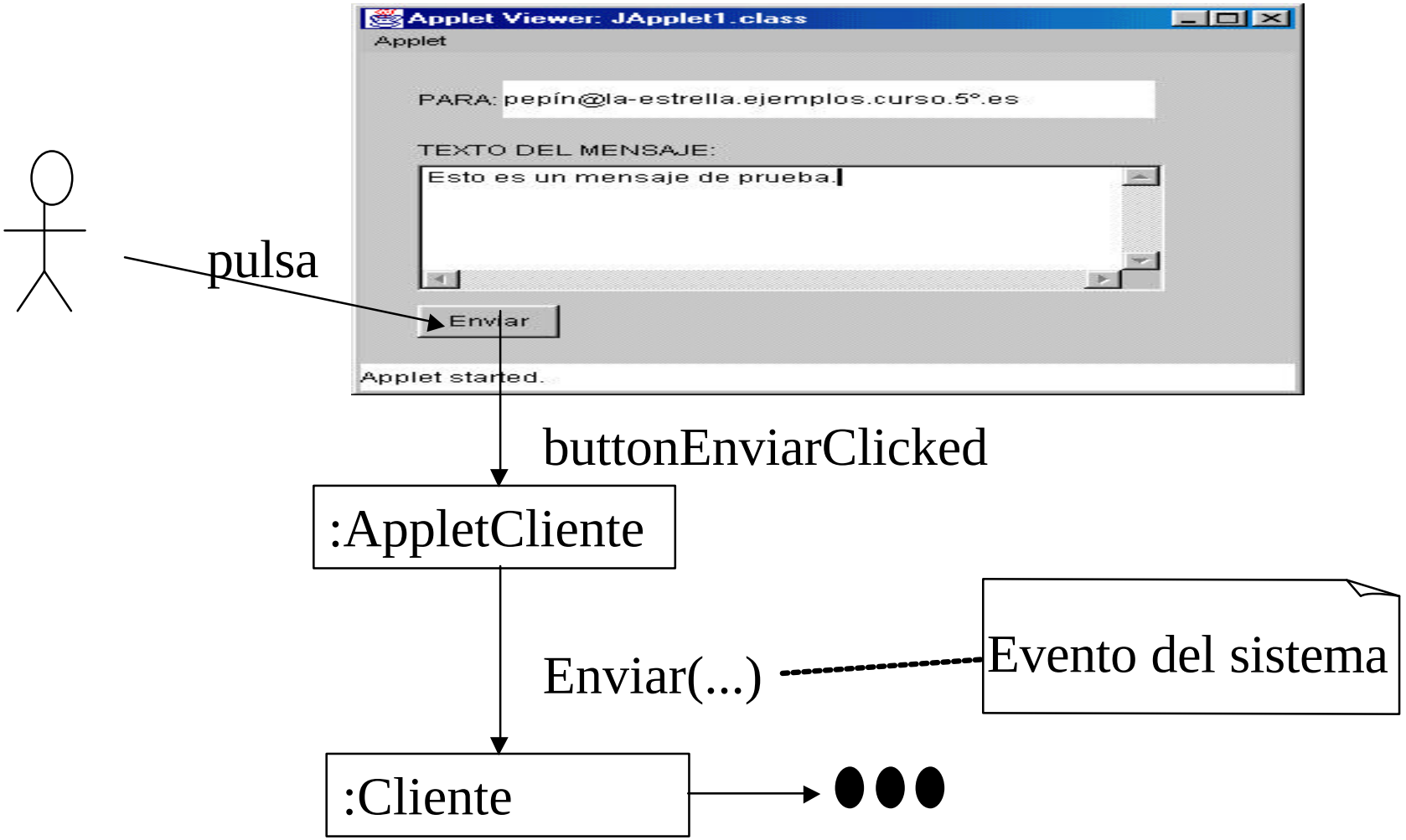
El patrón “Controlador” (II)

- Solución: *asignar la responsabilidad al “controlador”, que será una clase que:*
  - *Representa al sistema completo, a la organización... (controlador “fachada”)*
  - *Representa una parte activa del mundo real que desencadena de tal evento (controlador de rol)*
  - *Representa un manejador artificial de eventos (controlador de CdU)*

El patrón “Controlador” (III)

- Sugerencias:
  - Puede utilizarse un controlador por cada CdU, de manera que se encargue de controlar el estado del CdU, la secuencia de eventos, etc.
  - Una vez que el controlador recibe un evento, puede delegar sobre otros objetos para no verse sobrecargado (producirá, además, baja cohesión).

El patrón “Controlador” (y IV)



El patrón “Comando” (no es Grasp)

- Se utiliza para asignar responsabilidades de manejo de eventos en sistemas que procesan muchos tipos de eventos.