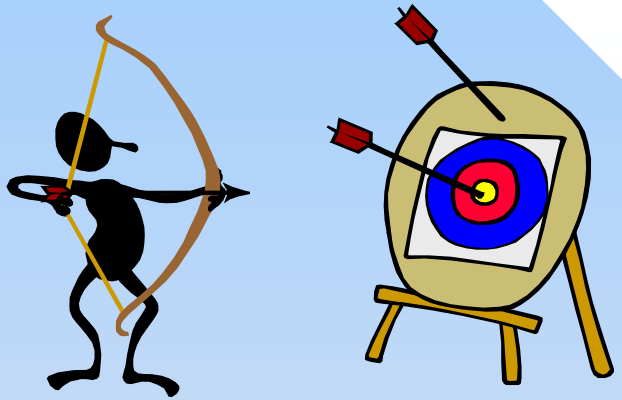
ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS (ADOO) CON UML

Alejandro Domínguez / Jorge Kashiwamoto / Enrique Torres

(Versión 4.0)

Curso impartido en la Fundación Arturo Rosenblueth y en UNITEC de 1998-2010

Objetivos generales



- Proporcionar el conocimiento de los principios inherentes en el análisis y diseño orientado a objetos
- Introducir el Lenguaje Unificado de Modelado (*Unified Modeling Language*: UML), el cual es el lenguaje estándar para el análisis y diseño orientado a objetos
- Utilizar UML para modelar un caso práctico de sistemas

Panorámica general del curso (1)

- El curso en 13 capítulos
  - Capítulo 1: Tratamiento de la complejidad y la orientación a objetos
  - Capítulo 2: Historia de la orientación a objetos
  - Capítulo 3: Conceptos de la orientación a objetos
  - Capítulo 4: Introducción a UML y al análisis y diseño orientado a objetos
  - Capítulo 5: Modelo de requerimientos (modelado de casos de uso)

Panorámica general del curso (2)

- El curso está compuesto por ...
  - Capítulo 6: Modelo de análisis (diagramas de clase)
  - Capítulo 7: Modelo de diseño (diagramas de secuencia)
  - Capítulo 8: Modelo de diseño (diagramas de estado)
  - Capítulo 9: Modelo de diseño (diagramas de colaboración)
  - Capítulo 10: Modelo de diseño (diagramas de actividad)
  - Capítulo 11: Modelo de implantación (diagramas de componentes)
  - Capítulo 12: Modelo de implantación (diagramas de "deployment")
  - Capítulo 13: Introducción al proceso unificado de desarrollo de software

Panorámica general del curso (2)

- El curso está compuesto por ...
  - Capítulo 13: Introducción al Proceso Unificado de Desarrollo de Software
    - RUP

Panorámica general del curso (3)

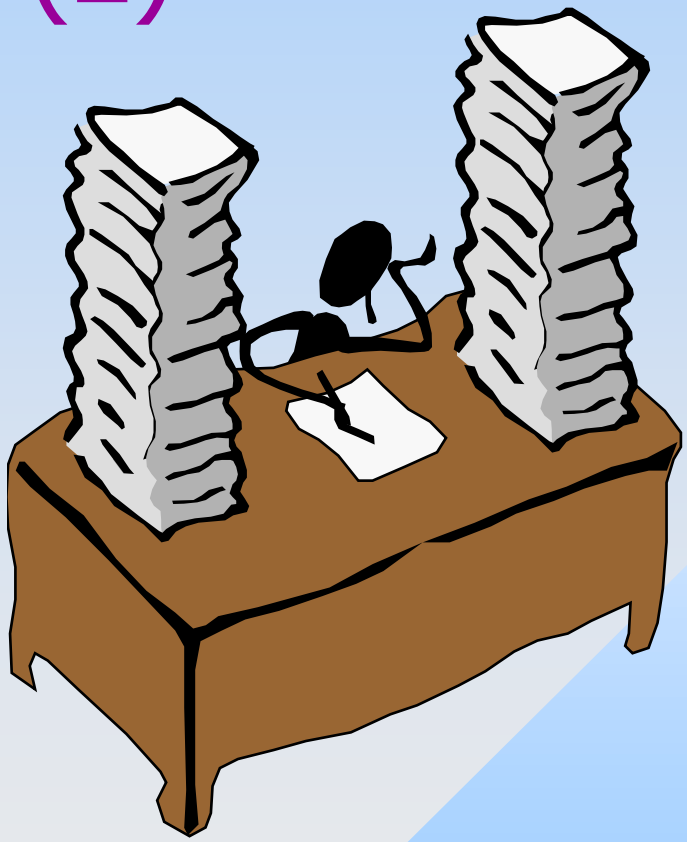
- Adicionalmente el alumno debe cubrir de forma autodidacta el aprendizaje de la herramienta *Rational Rose* o *Visio* de las partes 4 a 12
  - Se puede obtener una versión de evaluación de Rational Rose por 30 días en http://www.rational.com
- En clase se presentarán algunos ejemplos para facilitar la comprensión de la elaboración de diagramas.

Material de consulta (1)

- Alejandro Domínguez *Análisis y Diseño Orientado a Objetos con UML Versión 3.0* FAR
- Booch, G., J. Rumbaugh y I. Jacobson. *The unified modeling language user guide*. Addison-Wesley, 1999.
- Fowler, M. Y K. Scott. *UML gota a gota*. Pearson, 1999.
- Oestereich, B. *Developing software with UML*. Addison-Wesley, 1999.

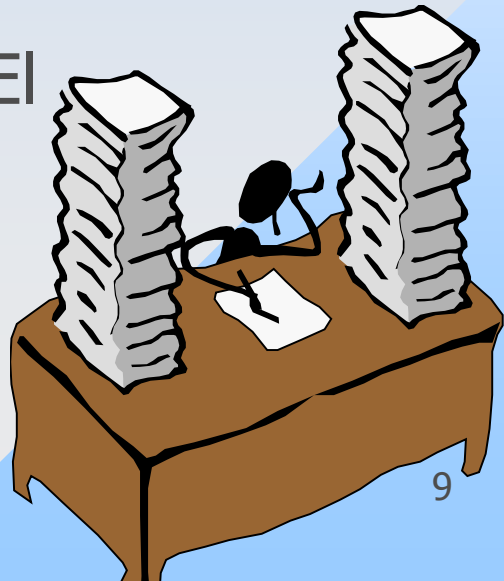
Material de consulta (2)

- Schneider, G. y J.P. Winters. *Applying use cases*. Addison-Wesley, 1999.
- Taylor, D.A. *Object technology*. Addison-Wesley, 1998.
- Rational Software, 1997. *UML notation guide. Version 1.1.* http://www.rational.com
- Rational Software, 1997. *UML semantics. Version 1.3.* http://www.rational.com



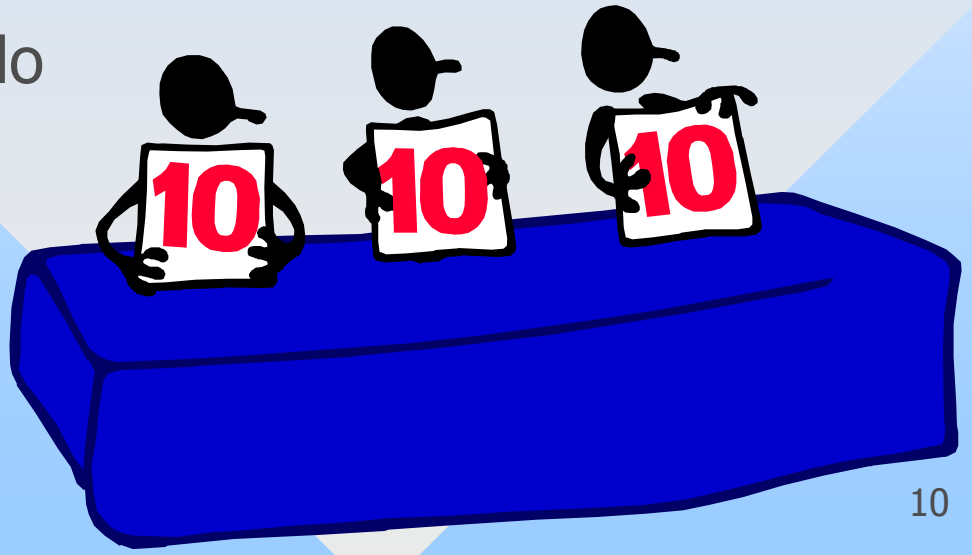
Material de consulta (3)

- Booch, G., J. Rumbaugh y I. Jacobson. *The unified modeling language reference guide*. Addison-Wesley, 1999.
- Larman, C. / *UML y Patrones* / Pearson / 1999
- Rumbaugh / *OMT* / Addison-Wesley
- Booch, G., J. Rumbaugh y I. Jacobson. El Proceso Unificado de Desarrollo de Software. Addison-Wesley, 1999.



Evaluación del curso

- 100% análisis y diseño de un caso práctico de sistemas que incluya:
  - Diagramas de casos de uso
  - Diagramas de clase
  - Diagramas de secuencia
  - Diagramas de estado



TRATAMIENTO DE LA COMPLEJIDAD Y LA ORIENTACIÓN A OBJETOS

Alejandro Domínguez

Panorámica general

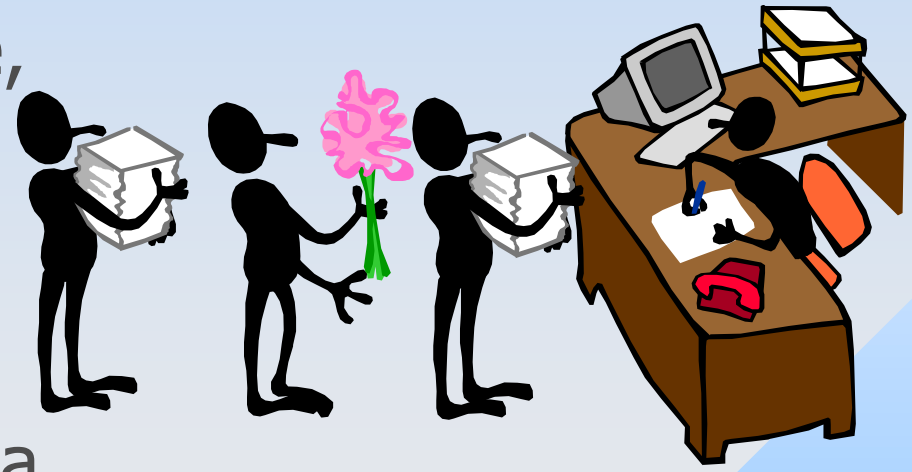
- Complejidad de los sistemas de software
- Tratamiento de la complejidad y descomposición
- Descomposición orientada a procesos y a objetos

Los sistemas

- Un sistema es:
  - Un grupo conexo y organizado de objetos
  - Un todo compuesto de partes organizadas acorde a algún esquema o plan
- Los sistemas tienen:
  - Fronteras
  - Entorno
  - Características propias
  - Propiedades emergentes

Sistemas complejos

- La complejidad de un sistema es una propiedad inherente, no accidental
- La interconexión de los sistemas es una fuente potencial para inconsistencias e incongruencias

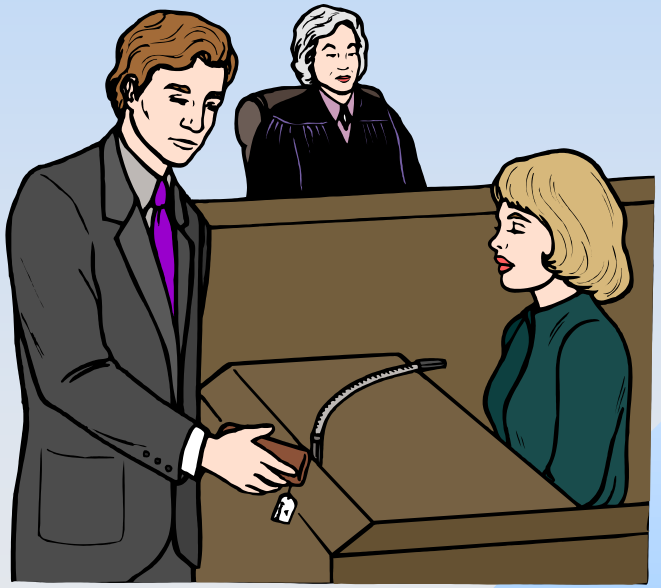


Origen de la complejidad de los sistemas

- Desarrollado por personas
- Desarrollado a través de un proceso largo y tedioso
- No se comprende el sistema en su totalidad
- Difícil de documentar y probar
- Potencialmente inconsistente o incompleto
- Sujeto a cambios
- La no existencia de leyes fundamentales para explicar los fenómenos y enfoques
- Existencia del "Límite de Miller"
  - El ser humano sólo puede manejar, procesar o mantener la pista de aproximadamente 7 objetos, entidades o conceptos a la vez

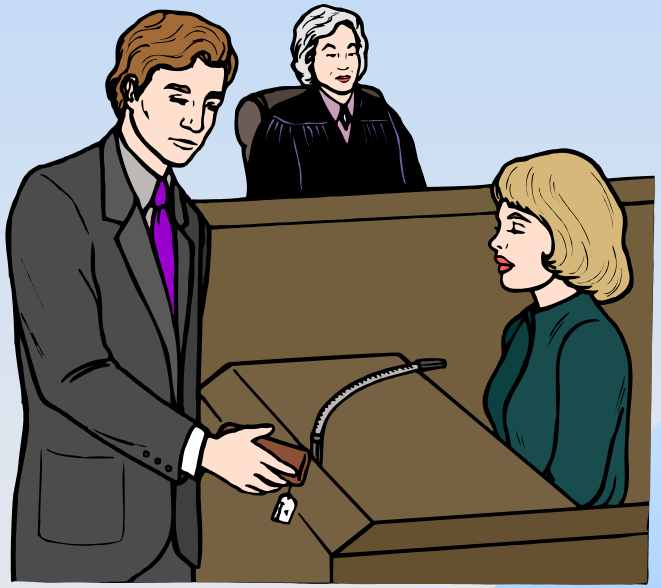
4 razones para la complejidad (Grady Booch)

- 1ª razón.- La naturaleza del dominio del problema:
  - Requerimientos complejos
  - Decaimiento de los sistemas
- 2ª razón.- Complejidad de los procesos:
  - Problemas en la administración y gestión
  - Necesidad de simplificación

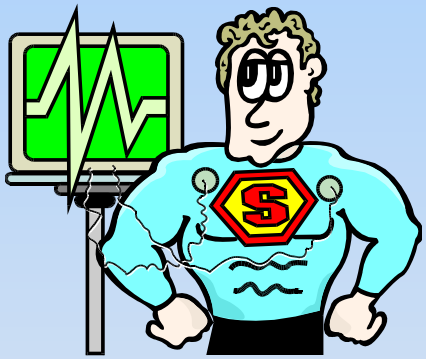


4 razones para la complejidad (Grady Booch)

- 3ª razón.- La flexibilidad de los sistemas es un arma de dos filos
- 4ª razón.- Caracterizar el comportamiento discreto de los sistemas:
  - Existencia de varios estados
  - Dificultad para explorar todos los estados

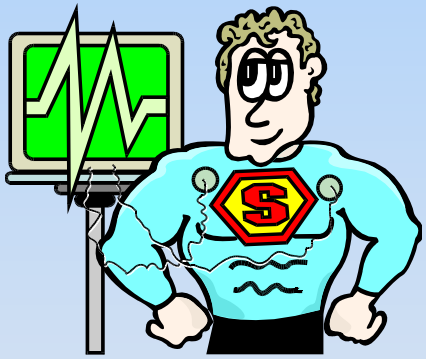


5 atributos de los sistemas complejos (Grady Booch)



- 1er atributo
  - Los subsistemas son jerárquicos e interactuan entre ellos
- 2° atributo
  - Los subsistemas se han definido arbitrariamente
- 3er atributo
  - El acoplamiento entre componentes del mismo subsistema es fuerte y el acoplamiento entre componentes de diferentes subsistemas es débil

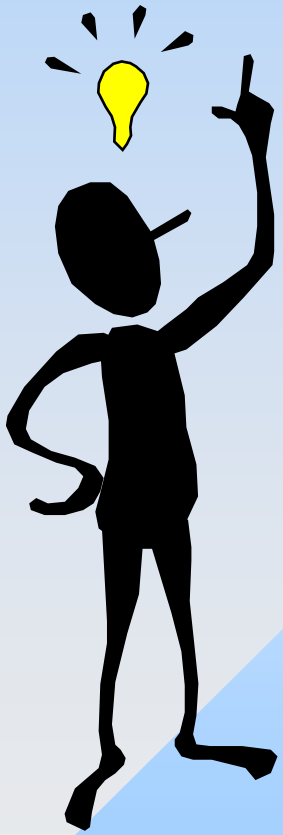
5 atributos de los sistemas complejos (Grady Booch)



- 4° atributo
  - Poseen una serie de subsistemas recurrentes que aparecen en diferentes combinaciones y arreglos
- 5° atributo
  - No siempre evolucionaron de sistemas simples a sistemas complejos

Tratamiento de la complejidad

- Principios que permitirán el tratamiento de la complejidad:
  - Abstracción
    - Ignorar los detalles no esenciales y construir un modelo ideal y generalizado
  - Jerarquía
    - Clasificar en grupos de abstracciones relacionadas para poder distinguir explícitamente las propiedades comunes y distintas
  - Descomposición
    - Orientada a procesos
    - Orientada a objetos

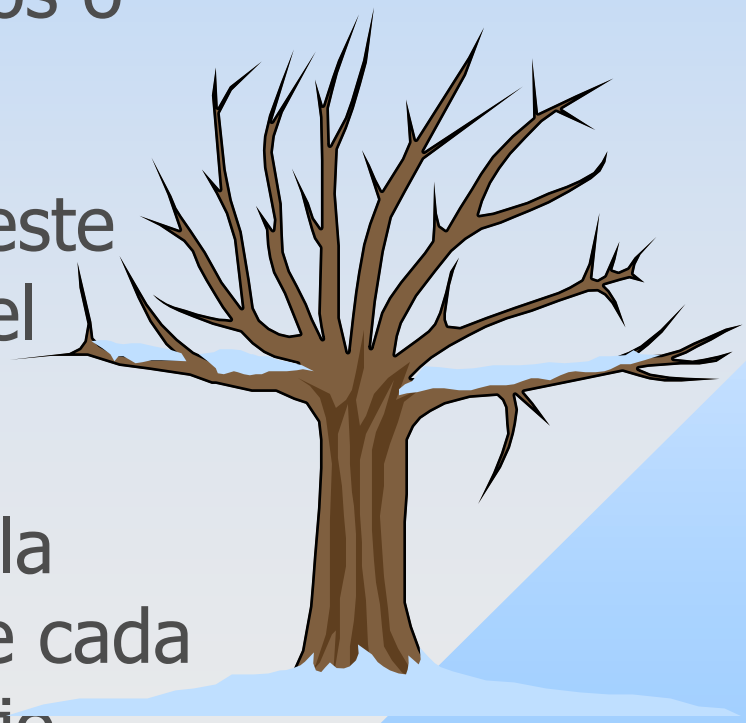


Descomposición

- La mejor técnica conocida desde tiempos inmemoriales para enfrentar la complejidad es: *divide et impera*
- Con esta técnica se solventa la restricción impuesta por el límite de Miller

Descomposición orientada a procesos (OP)

- Se aplica para descomponer un problema en pequeños pasos o procesos
- La unidad fundamental de este tipo de descomposición es el "subproceso"
- El proceso resultante toma la forma de un árbol en el que cada subproceso realiza su trabajo, llamando a otro subproceso

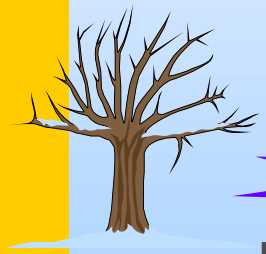


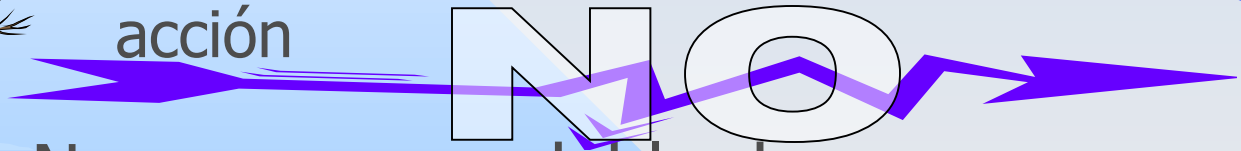
Descomposición orientada a objetos (OO)

- El sistema se visualiza como un conjunto de objetos autónomos, que al colaborar entre si muestran cierta conducta
- Los procesos no existen de manera independiente, están asociados a las objetos
- Cada objeto exhibe un comportamiento propio bien definido, y además modela alguna entidad del mundo real

Descomposición OP vs OO

- ¿Cuál es la forma correcta de descomponer un sistema complejo, OP u OO?
- Las dos formas son importantes
  - La OP muestra el orden de los eventos
  - La OO enfatiza las entidades que causan la acción

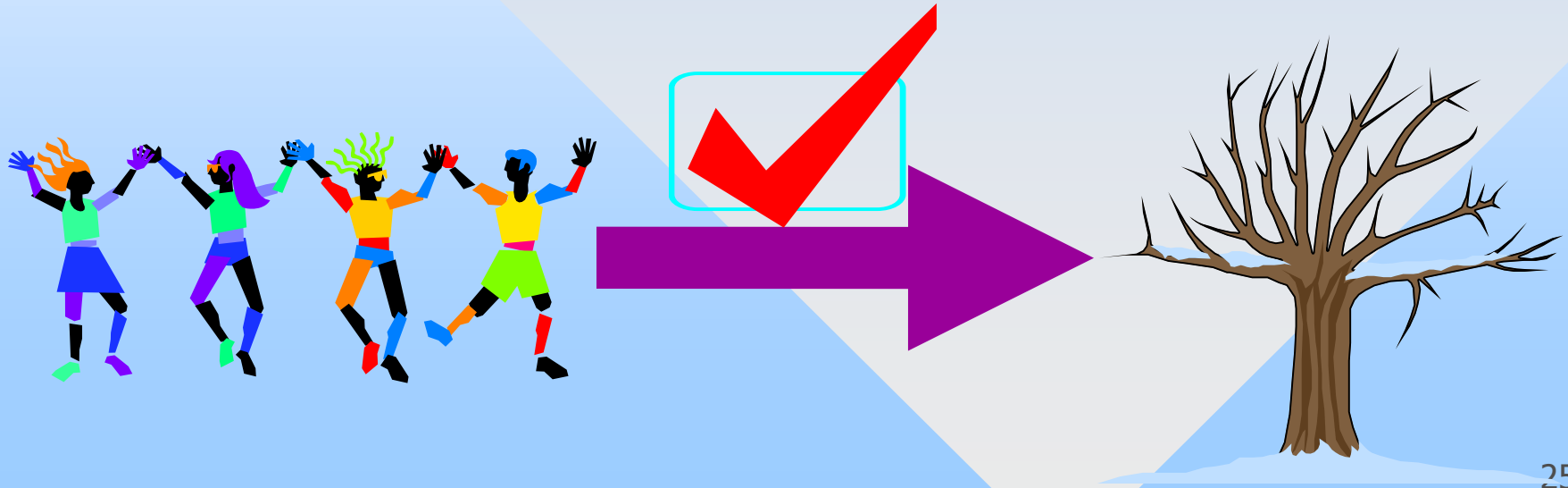




- No es recomendable descomponer un sistema complejo de ambas formas simultáneamente debido a que se incrementa la complejidad

Recomendación para el tratamiento de la complejidad

- Primero aplicar un descomposición OO, la cual servirá como marco de referencia para continuar con una descomposición OP



HISTORIA DE LA ORIENTACIÓN A OBJETOS

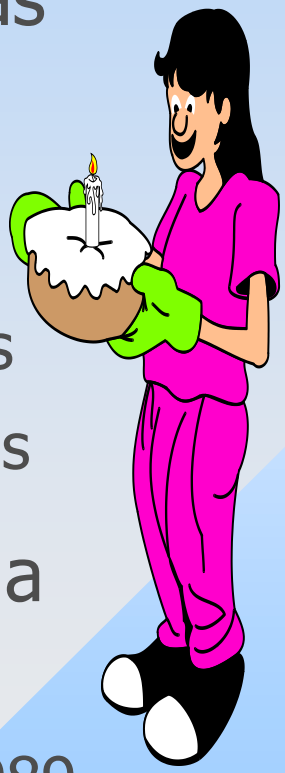
Alejandro Domínguez

Panorámica general

- La vida media de las tecnologías de IS
- Aspectos históricos de la orientación a objetos(OO)
- La OO en lo 1990’s
- Lo que se ha aprendido de la OO

15 años de fama

- La experiencia nos dice que las tecnologías de ingeniería de software (IS) más populares tienen el siguiente comportamiento:
  - Toman algún tiempo en llegar a ser populares
  - Tienen un periodo de florecimiento de 15 años
- Un ejemplo es la metodología estructurada
  - Empezó a ser popular entre de 1973-1975
  - Alcanzó su pico de popularidad entre 1985-1989



La "vida media" para las tecnologías de IS

- La vida media para las tecnologías indica el punto medio en su ciclo de vida
- La vida media en la orientación a objetos significa que:
  - Se ha utilizado esta tecnología por algún tiempo en proyectos reales
  - Será remplazada por otras tecnologías dentro de un periodo de 7 a 10 años

Aspectos históricos (1)

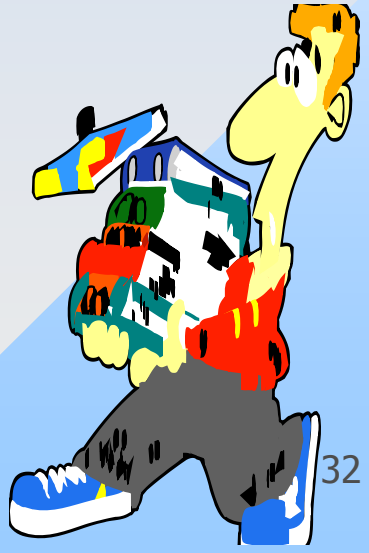
- 1963:
  - **I. Sutherland** diseña **Sketchpad** en el cual se presentan gráficas e interfaces gráficas de usuario OO; así como clases (*masters*) e instancias
- 1966:
  - Se introduce al mercado **Simula**, el primer lenguaje de programación OO
- Finales de 1960's:
  - **Alan Kay** trabaja en la máquina denominada **FLEX**, la precursora de **Dynabook**, la **Xerox Star**, y la **Apple Macintosh**

Aspectos históricos (2)

- 1970:
  - El término "orientado a objetos" se introduce en la literatura. Muchas personas se lo acreditan a Alan Kay
- 1972:
  - Se desarrolla la primera versión de **Smalltalk**
- 1980:
  - Grady Booch introduce el "diseño orientado objetos"
  - Se introducen las primeras versiones de "hardware OO"

Aspectos históricos (3)

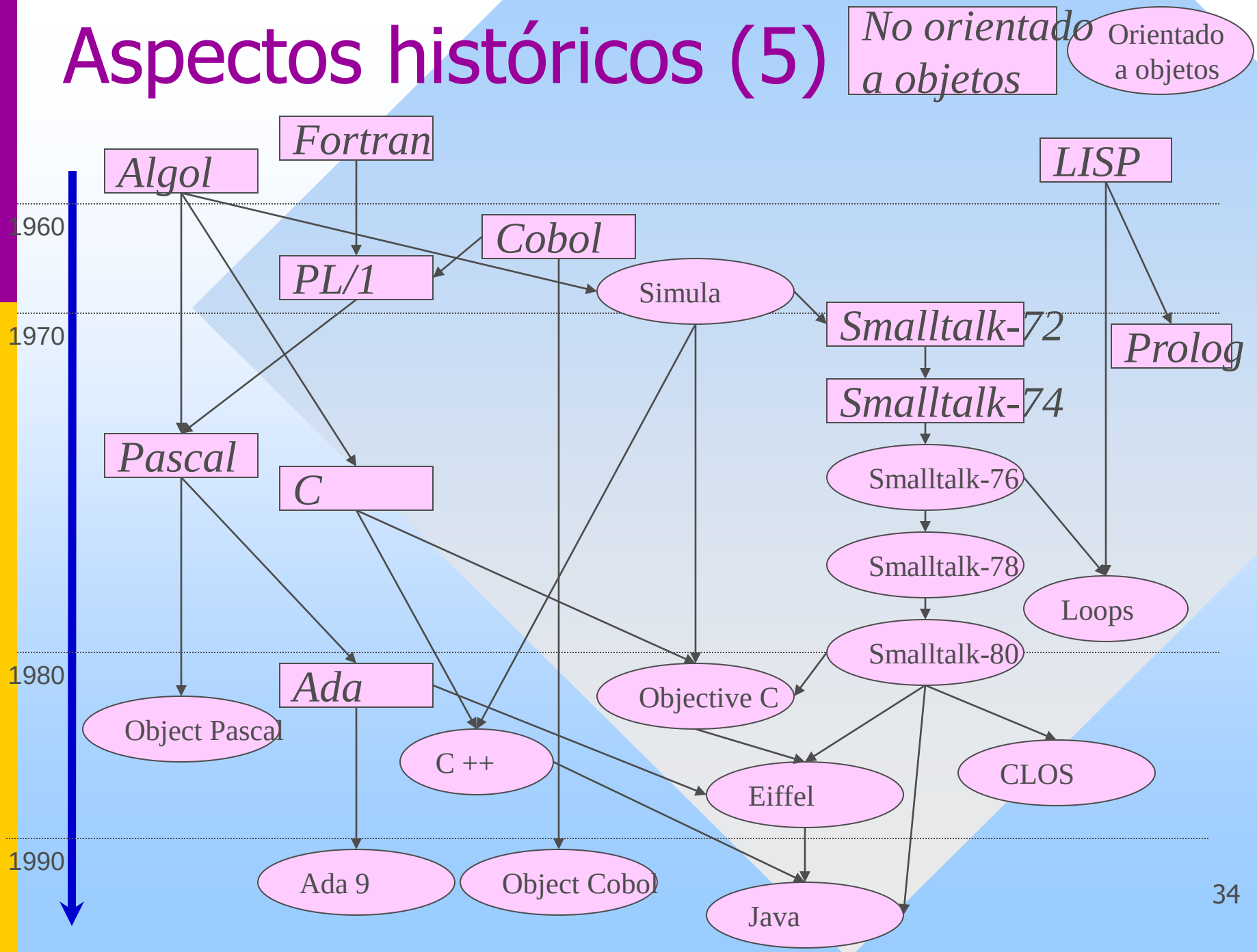
- 1985:
  - aparecen las bases de datos OO
  - aparecen la bibliotecas de clases OO
- 1986:
  - Se introducen las primeras versiones de "análisis OO" y de "análisis de requerimientos OO"
- 1988:
  - "análisis del dominio OO" aparece en la literatura



Aspectos históricos (4)

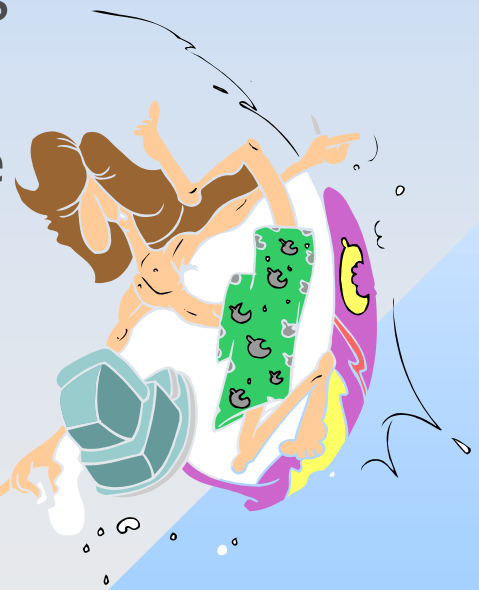
- 1980's:
  - Se desarrollan un gran cantidad de aplicaciones OO
    - Estas aplicaciones son principalmente "aplicaciones de tiempo real"
    - Literalmente, se producen millones de líneas de código OO
- Finales de 1980's:
  - Aparece un gran número de lenguajes de programación OO; incluyendo: C++, Eiffel, CLOS, etc.

Aspectos históricos (5)



Mediados de los 1990's (1)

- La MOO empieza a mostrar signos de "vida media":
  - Literalmente cientos de millones de líneas de código fuente OO se han escrito
    - Algunos productos OO contienen alrededor de 2,000,000 de líneas de código
  - La tecnología de BDOO tiene un éxito sin precedente
    - La Object Management Group (OMG) hace un esfuerzo por estandarizar la tecnología de BDOO
    - El mercado de los sistemas de bases de datos OO se duplica cada año



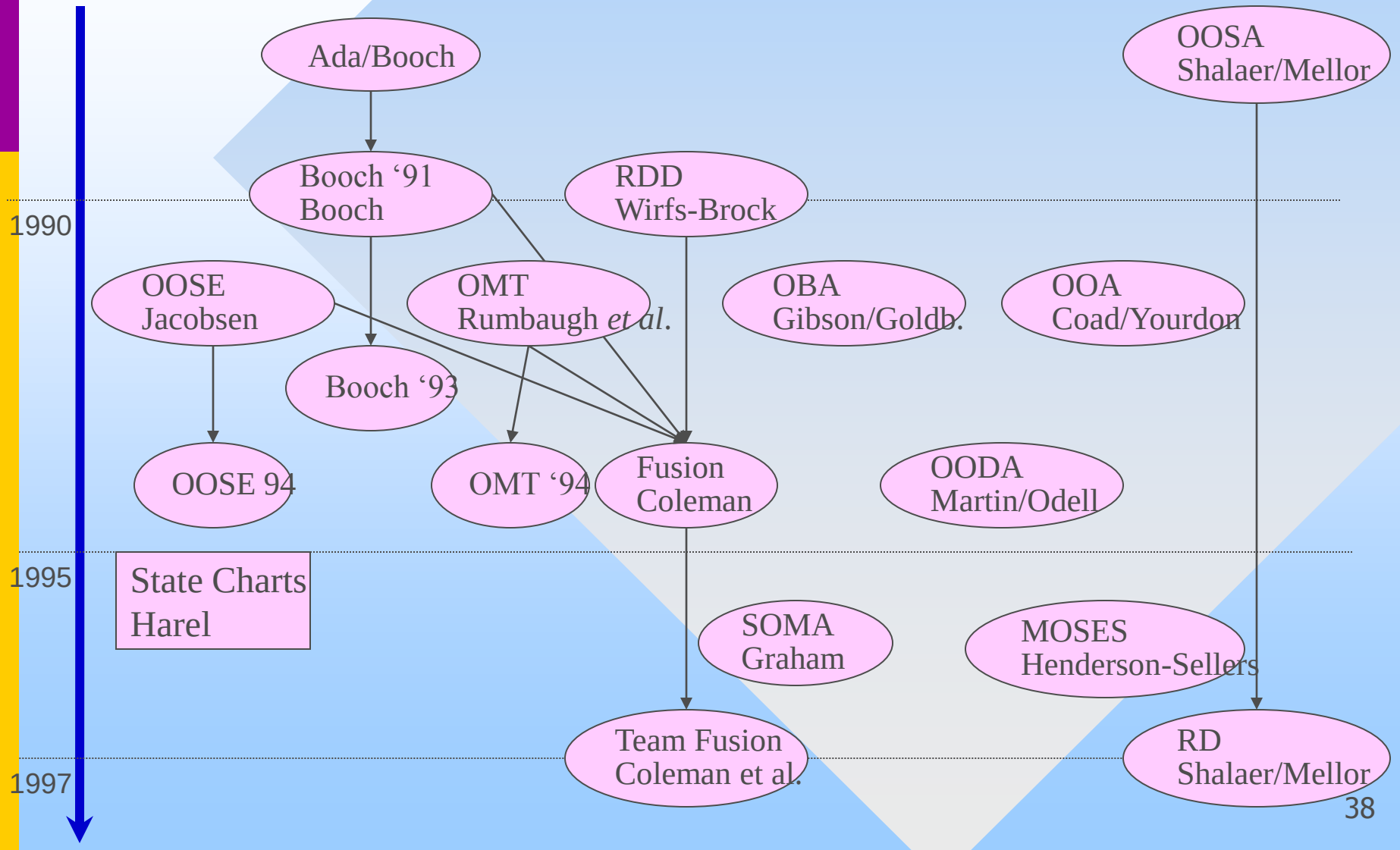
Mediados de los 1990's (2)

- La MOO empieza a mostrar signos de "vida media":
  - Los desarrolladores de software pueden elegir entre un gran número de herramientas CASE OO
    - Object International's Object Tool
    - Rational's Rose
    - Protosoft's Paradigm Plus
  - Existen muchos enfoques de desarrollo de software OO: Booch, Rumbaugh, Coad, Wirfs-Brock, Berard, Fusion, Shlaer-Mellor, ...
    - Alrededor de 30 enfoques son populares

Mediados de los 1990's (3)

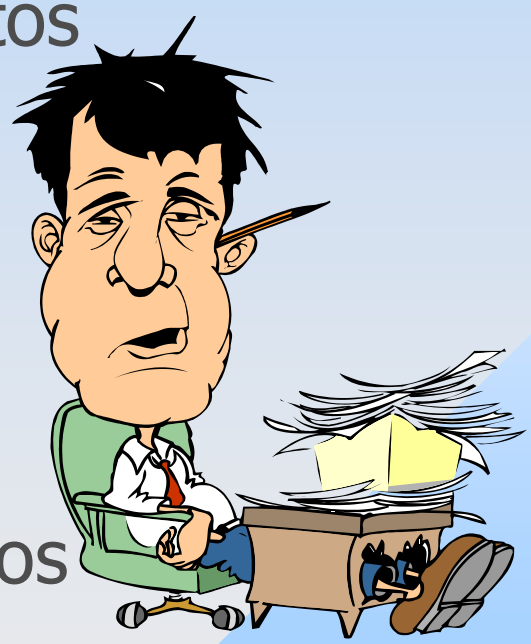
- La MOO empieza a mostrar signos de "vida media":
  - El término "orientado a objetos" aparece de forma regular en las publicaciones de negocios: The Wall Street Journal, Business Week, ...
  - La comunidad de administración de los SI empieza a utilizar la tecnología OO

Mediados de los 1990's (4)



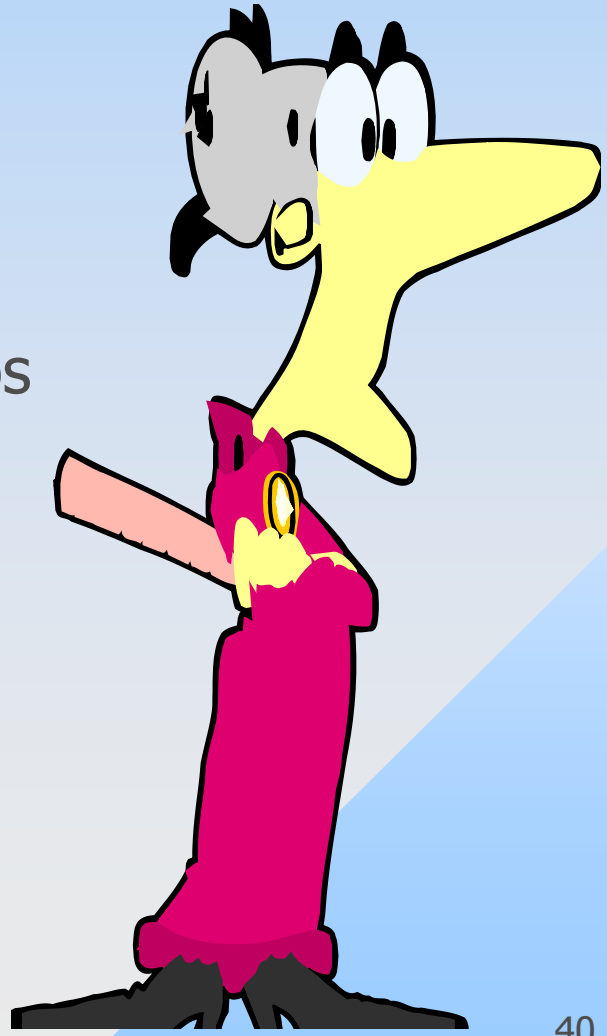
A finales de los 1990's

- Existe una tendencia a estandarizar las tecnologías orientadas a objetos
  - *Unified modeling language* (UML)
  - *Business objects*
- Existe un crecimiento significante (y explosivo) de tecnologías basadas en la orientación a objetos
  - programación basada en componentes
  - programación basada en agentes



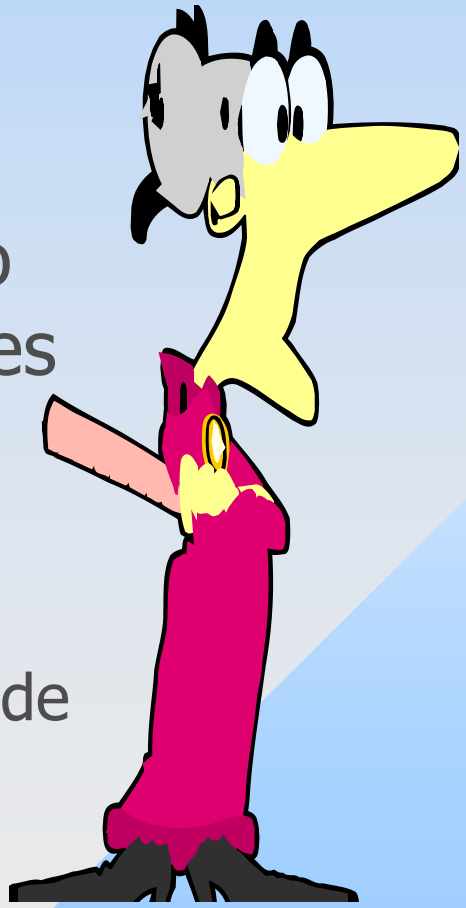
Lo que se ha aprendido (1)

- La tecnología OO es una oportunidad, no una garantía
  - Los proyectos orientados a objetos pueden fallar
- La tecnología OO es vasta
  - Más amplia que lo indicado en los lenguajes de programación OO



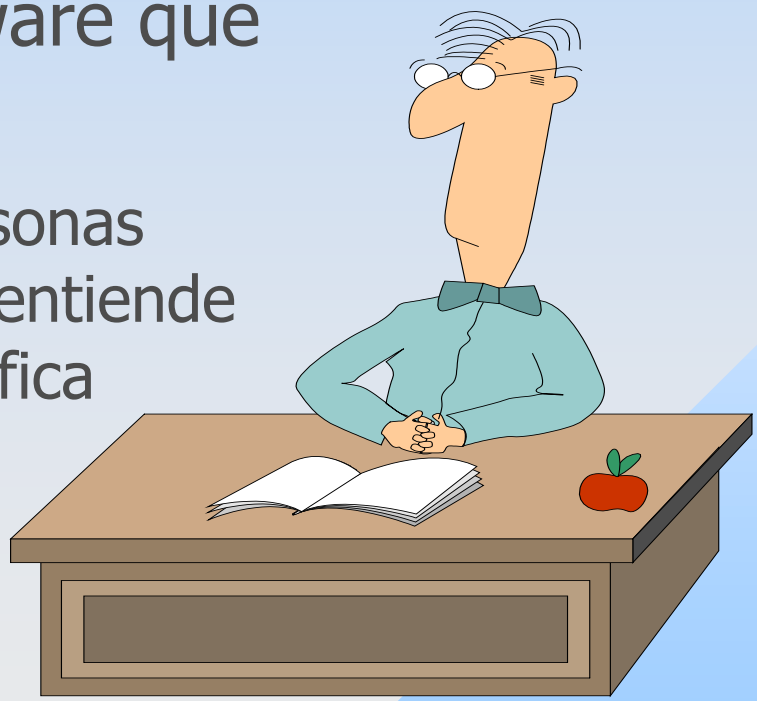
Lo que se ha aprendido (2)

- La evidencia empírica muestra que, cuando se compara con las mismas aplicaciones desarrolladas utilizando enfoques tradicionales, las soluciones OO
  - Son más pequeñas
  - Son menos complejas
  - Son más apropiadas para aplicaciones de tiempo real
  - Toman menos tiempo en desarrollarse



Lo que se ha aprendido (3)

- La tecnología OO hace mas énfasis en la reusabilidad del software que los métodos tradicionales
  - Sin embargo muy pocas personas dentro de la comunidad OO entiende lo que verdaderamente significa reusabilidad



Lo que se ha aprendido (4)

- La tecnología OO tiene impacto en los dominios de:
  - Las practicas administrativas
  - Las metodologías del ciclo de vida
  - La selección de herramientas
  - El almacenamiento persistente
  - Los lenguajes de programación

CONCEPTOS DE LA ORIENTACIÓN A OBJETOS

Jorge Kashiwamoto / Alejandro Domínguez

Panorámica general

- La orientación a objetos
- Objetos y abstracción
- Clasificación y clases
- Características de los Objetos
- Relaciones
- Agregación
- Modularidad
- Encapsulamiento
- Generalización y especialización
- Polimorfismo
- Herencia y herencia múltiple
- Conclusiones

¿Qué es la Orientación a Objetos?

Paradigma

PARADIGMA	ASPECTOS ESENCIALES QUE MODELA
Procedural	Algoritmos
Funcional	Metas u objetivos en un cálculo de predicado
Restricciones	Relaciones invariantes
Objetos	Clases y objetos

Paradigma de la ORIENTACIÓN A OBJETOS

- Marco conceptual propio
  - Análisis de problemas
  - Enfoque particular del problema.
- Planteamiento más claro
  - Determinados problemas
  - Determinado marco conceptual
- Apropiado para el desarrollo de aplicaciones comúnes
- Donde el análisis de la complejidad juega el papel más importante

Derivados de la OO

- Tecnologías
- Metodologías
- Programación
- Lenguajes
- Bases de Datos
- Etc.

Motivaciones de la OO

- Tratar o abordar dominios de problemas más amplios y complejos
- Mejorar la interacción entre los usuarios y los analistas.
- Incrementar la consistencia entre análisis, diseño y programación
- Representar explícitamente los aspectos comunes dentro de una clase o familia de entidades.
- Construir especificaciones fáciles de cambiar o adaptar
- Reusar resultados previos de análisis, diseño y programación
- Proveer una representación subyacente consistente para el análisis, el diseño y la programación

Orientación a Objetos

- El software se organiza como una colección de objetos discretos que contienen tanto estructuras de datos como comportamiento
  - Rumbaugh
- Los objetos reúnen las características esenciales que los definen.

La orientación a objetos (OO)

- Significa notaciones, técnicas y métodos para producir varios modelos de un sistema en diferentes niveles de abstracción y granularidad basados en los principios de:
  - Objetos
  - Clases de objetos
  - Encapsulamiento
  - Herencia
  - Polimorfismo

Definición de objeto

- Un objeto es:
  - Diccionario Larousse: Objeto (del bajo latín *objectum* < lat.)
    - Cosa material y concreta, por lo regular de dimensiones reducidas
    - Término con lo que se denota aquello sobre lo que recae la acción expresada por el verbo
  - "The 3 amigos":
    - Una manifestación concreta de una abstracción
    - Una entidad con una frontera e identidad bien definidas que encapsula estado y comportamiento
    - Una instancia de una clase

OBJETO

- Es una entidad que retiene una cierta información abstracta y sabe como realizar ciertas operaciones sobre esta información
  - Wirfs-Brock
- OBJETO =
  - ATRIBUTOS +
  - COMPORTAMIENTO
- Entidad discreta con límites e identidad bien delimitadas que encapsula el estado y el comportamiento. Instancia de una clase.
  - UML

Ejemplos de objetos

- En una empresa procesadora de alimentos
  - Objetos pasivos:
    - Un saco de lentejas
    - Un paquete de café
    - Factura 895 enviada a alguna empresa
  - Objetos activos:
    - Camioneta placas "1234AB"
    - La máquina de fax de la empresa
  - Agentes humanos
    - Zoila Baca del Corral (Secretaria ejecutiva)
    - Armando B. Roncas (Guardia de seguridad)
  - Objetos estructurales
    - Departamento de ventas

Abstracción

- **"La abstracción surge desde un reconocimiento de similaridades entre ciertos objetos, situaciones o procesos en el mundo real, y la decisión de concentrar la atención sobre estas similaridades e ignorar por el momento las diferencias".[Hoare]**
- **"La abstracción es una descripción simplificada, o especificación, de un sistema que enfatiza algunos de los detalles o propiedades del sistema mientras suprime otras. Una buena abstracción es una que enfatiza detalles que son significativos al lector o usuario y suprime detalles que son, al menos por el momento, inmateriales o que distraen la atención". [Shaw]**

Abstracción

- **"Un concepto califica como una abstracción sólo si este puede ser descrito, comprendido y analizado independientemente del mecanismo que eventualmente será usado para realizar éste". [Berzins, Gray y Naumann]**
- **"Una abstracción denota las características esenciales de un objeto que distinguen a éste de todos los otros tipos de objetos y de esta forma proporciona límites conceptuales definidos sucintamente, relativos a la perspectiva del observador". [Booch]**

Abstracción

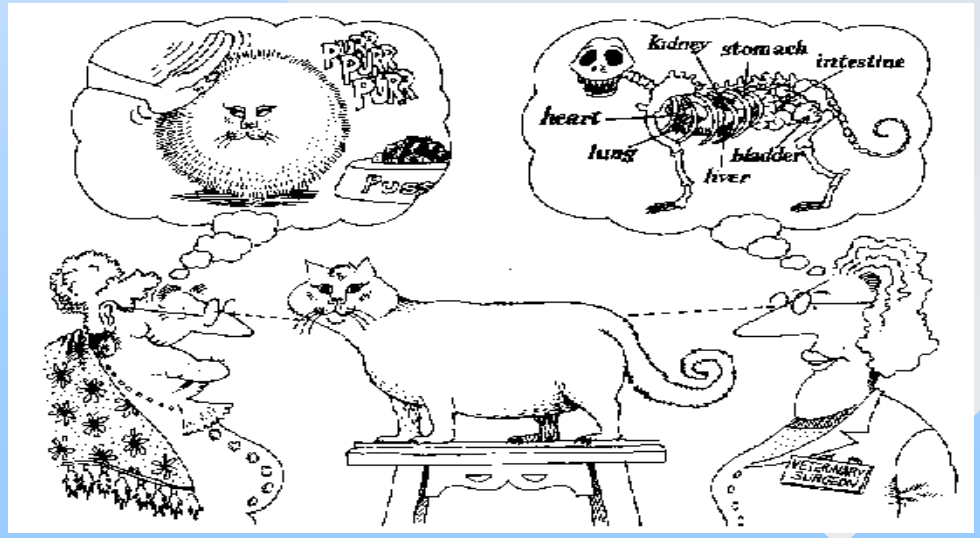
- **"Abstracción es el principio de ignorar aquellos aspectos de un sujeto que no son relevantes al propósito actual, con el objetivo de concentrar la atención completamente en aquellos que lo son". [Oxford University]**
- **"La abstracción de datos es el principio de definir un tipo de dato en términos de las operaciones que se aplican a objetos de este tipo, con la restricción de que los valores de tales objetos pueden ser modificados y observados sólo mediante el uso de estas operaciones". [Oxford University]**

Abstracción (Resumen 1/2)

- Forma fundamental en que los seres humanos se enfrentan con la complejidad de los problemas del mundo circundante.
- Una abstracción denota las características esenciales de un objeto, distinguiéndolo de otros tipos de objetos y obviando características irrelevantes a los intereses del observador que hace la abstracción

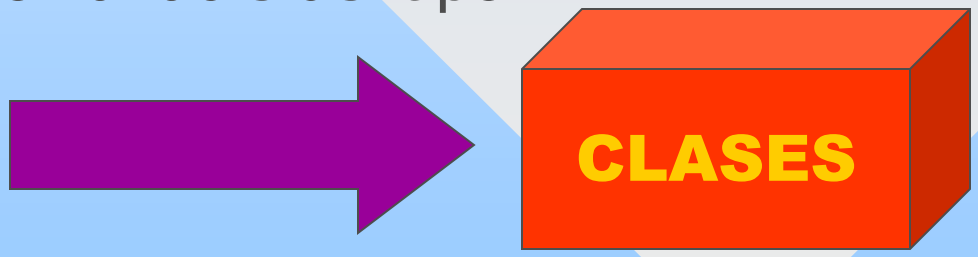
Abstracción (Resumen 2/2)

- Según Booch
  - Una abstracción denota las características esenciales de un objeto que lo distingue de todos los demás y, por lo tanto, proporciona fronteras conceptuales definidas, relativas a la perspectiva del observador
- Lo que un observador concibe no necesariamente lo concibe un observador diferente

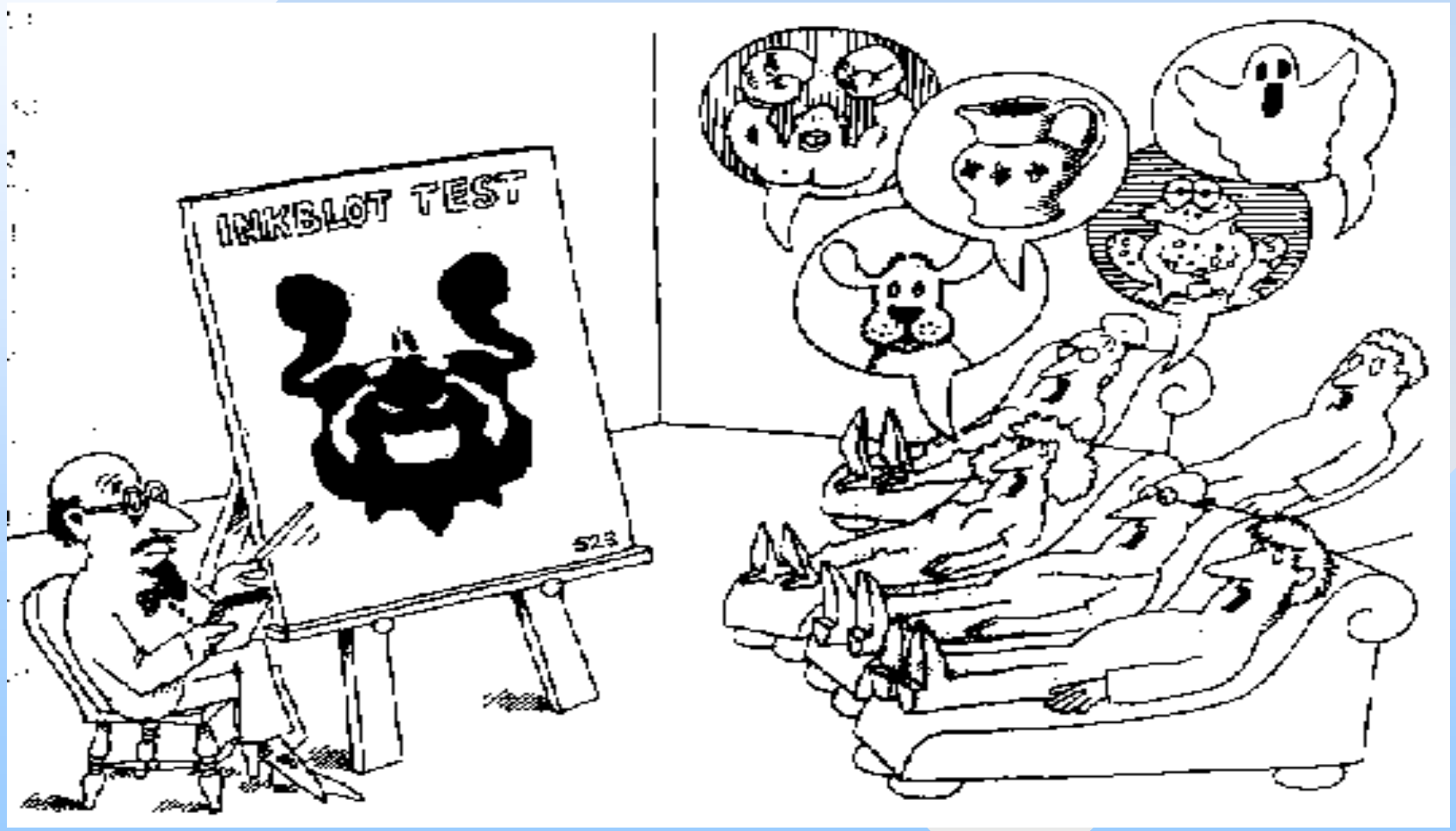


Clasificación

- Los objetos con iguales estructuras y comportamiento (operaciones) se agrupan para formar una clase que los describe.
- En este sentido el concepto de clase nos permite dividir y clasificar los objetos de nuestro problema.
- Se dice que la clase en tanto describe funciona como el "tipo" y el objeto en tanto es descrito y puede tener una existencia concreta, como la instancia o variable del tipo.



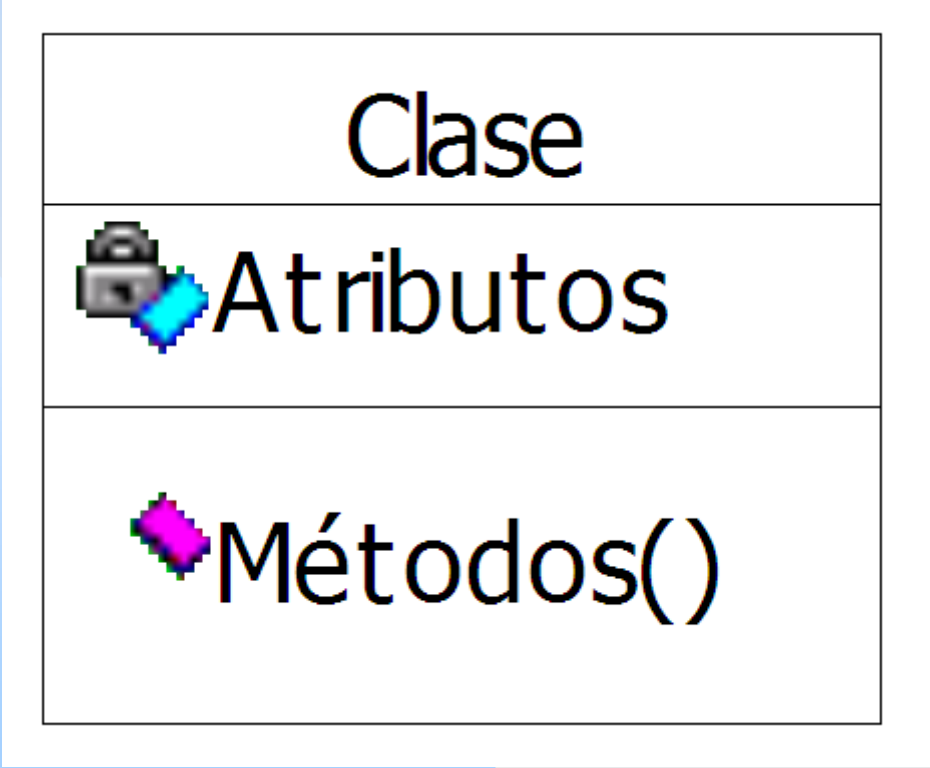
Dificultades de la clasificación



CLASE

- Es una colección de objetos los cuales comparten las mismas características y comportamientos.
  - Wirfs-Brock
- Descriptor para un conjunto de objetos que comparten los mismos atributos, operaciones, métodos, relaciones y comportamiento. Concepto del sistema modelado.
  - UML

Representación de clase



Tipo de dato abstracto
Abstract Data Type (ADT)

- Un tipo de dato abstracto es una caracterización precisa de una entidad en termino de atributos y operaciones, la cual brinda una interfaz y una implementación.
- Su interfaz es el medio de comunicación con el exterior, donde se describe la forma en que deben ser invocada cada una de las operaciones, mientras que su implementación consiste en la representación interna que tengan cada uno de los atributos, así como la forma concreta en que cada una de las operaciones actúa sobre dichos atributos. La implementación es una parte oculta al usuario del ADT, un mismo ADT puede tener varias implementaciones pero una única interface.

Ejemplo de Pila (abstracción)

programación en C muy primitiva

```
typedef int T;
# define Max_Stack 100
T stack[Max_Stack];
int top=0;
T item;

/* Interface para el manejo de la pila */
int create(int size);
int destroy(void);
void push(T new_item);
void pop(T *old_top);
void top(T *cur_top);
int is_empty(void);
int is_full(void);
```

1

programación en C con abstracción

```
typedef int T;
typedef struct{
  int top, size;
  T *stack;
}stack;

int stack_create(stack *s, int size);
void stack_destroy(stack *s);
void stack_push(stack *s, T item );
void stack_pop(stack *s, T *item);
int stack_is_empty(stack *s);
int stack_is_full(stack *s);
```

2

Ejemplo de Pila (ADT)

Programación en C++ con abstracción en Objectos

```
typedef int T;

class stack{
public: //interface
stack(int size);
~stack(void);
void push(const T &item);
void pop(T &item);
int is_empty(void) const;
int is_full(void) const;
private: //implementación
int top_, size_;
T *stack;
}
stack s1(10);  s1.push(473);
```

3

Abstracción y genericidad usando POO

```
template <class T>
class stack{
public: //interface
stack(int size);
~stack(void);
void push(const T &item);
void pop(T &item);
int is_empty(void) const;
int is_full(void) const;
private: //implementación
int top_, size_;
T *stack;
}
stack<int> s1(10);  s1.push(10);
stack<empleado> s2(20);
s2.push(juan);
```

Características de un objeto

- **Abstracción / Clasificación**
- **Encapsulamiento**
- **Herencia**
- **Polimorifismo**
- **Identidad**
- **Reusabilidad**
- **Modularidad**
- **Extensibilidad**
- **Persistencia**
- **Concurrencia**
- **Estado**
- **Comportamiento**

Características básicas de los objetos

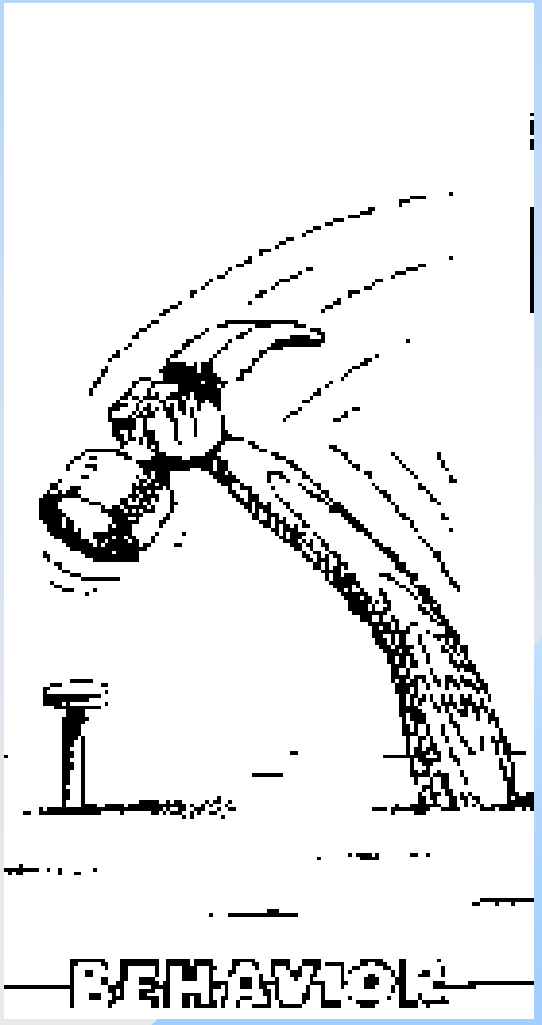
- Un objeto tiene
  - Estado
  - Comportamiento
  - Identidad

Estado

- Una condición o situación durante la vida de un objeto que satisface alguna condición, lleva a cabo alguna actividad, o espera algún evento
  - **Transición**
    - Cambio de estado.
  - **Evento**
    - Un suceso u ocurrencia de un fenómeno que lleva a una transición.

Comportamiento

- Los efectos observables de un evento, incluyendo sus resultados



Comportamiento: Implantación

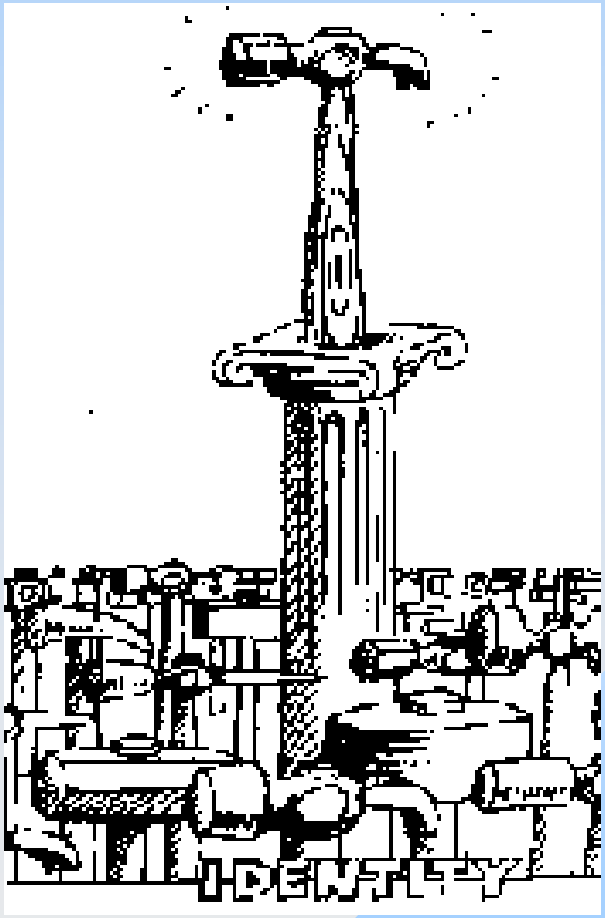
- **Operación**
  - Es una función de transformación sobre los atributos del objeto (no necesariamente cambiándolos). Los tipos más comunes de operación sobre un objeto son: *Modificación, Selección, Iteración.*
- **Método**
  - Es la implementación específica de una operación. Algunos especialistas utilizan los términos Operación y Método de manera intercambiable.

Comportamiento: Tipo de Objetos

- **Objetos Activos y Objetos Pasivos**
  - Los objetos activos son aquellos que dirigen su propio hilo de control. Los objetos pasivos sólo pueden ejecutar transiciones cuando se actúa sobre ellos. Dado que los objetos activos son autónomos, ellos son la raíz del control en un sistema. La existencia de hilos múltiples de control en un programa (concurrencia) sólo puede lograrse con múltiples objetos activos, mientras que los sistemas secuenciales sólo tienen un objeto activo a la vez.
- **Objeto Cliente y Objeto Servidor**
  - El objeto servidor responde a una solicitud de servicio (mensaje) del objeto cliente. Se establece una relación de "contrato cliente-servidor". Según este contrato el servidor debe cumplir las responsabilidades que se le exigen en el mismo.

Identidad

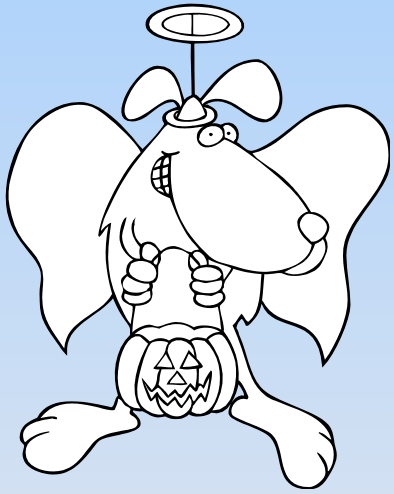
- Propiedad que permite distinguirlo de todos los demás objetos existentes



Identidad

- Los datos están cuantificados en entidades discretas y distinguibles, con fronteras bien definidas, denominadas objetos.
- Cada objeto tendrá una existencia propia, independientemente del valor de sus atributos.
- No es diferenciar los objetos por su nombre, sino ellos mismos, por el hecho de existir.
- Todo objeto contiene una referencia a si mismo.
- Ejemplo
  - clasedefinida a,b
  - a <> b

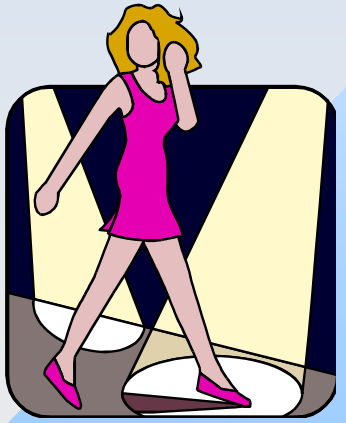
Propiedades de los objetos (1)



- Atributos
  - Son la estructura de los objetos: sus componentes y la información o datos contenidos en él
    - Son las características o propiedades intrínsecas o internas de los objetos
  - Son pares del tipo "nombre: valor"
    - El nombre debe reflejar la semántica de la característica o propiedad que representa
    - El nombre se utiliza para accesar el valor de la característica del objeto
    - El valor de los atributos puede cambiar en el tiempo, pero este no es el caso del nombre
  - Determinan el estado del objeto

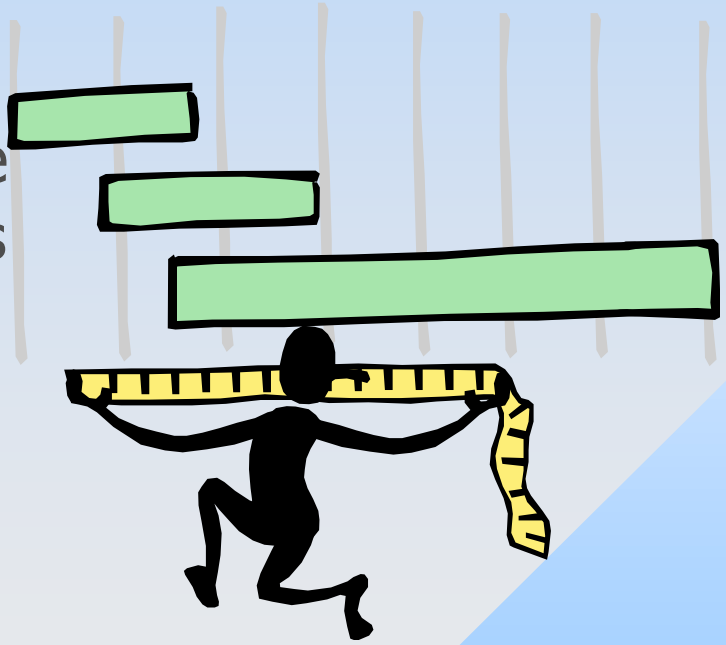
Propiedades de los objetos (2)

- Operaciones
  - Son servicios que pueden realizar los objetos cuando se les envía un mensaje
  - Determinan el comportamiento de los objetos
  - Algunas veces se utilizan las palabras 'servicios' o 'métodos' o erróneamente 'procedimientos' o 'funciones'



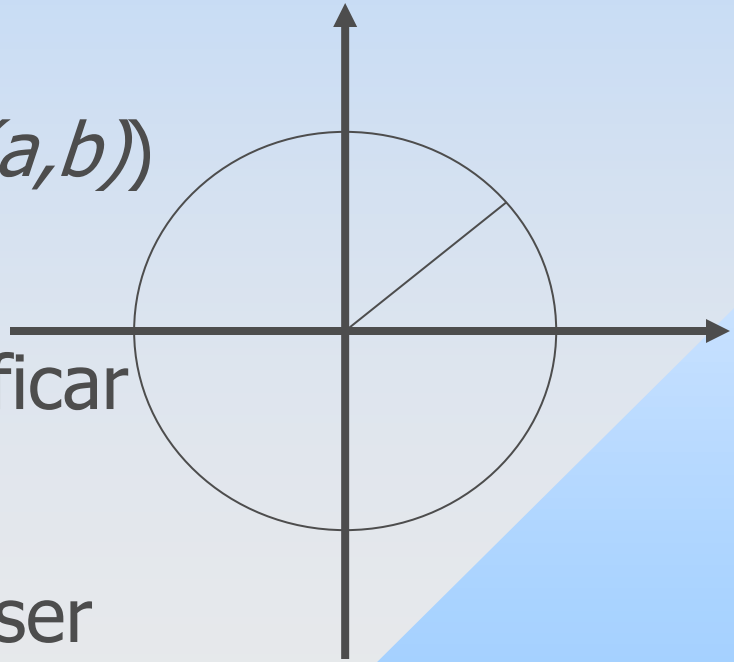
Propiedades de los objetos (3)

- Restricciones
  - Son condiciones, requerimientos y reglas que deben satisfacer los objetos
  - Restringen las propiedades y estados de los objetos
  - Regularmente están determinadas por los valores que pueden tomar los atributos



Ejemplo: un circulo

- Nombre: circulo
- Atributos: radio (*radio:r*) y posición del centro (*centro:(a,b)*)
- Operaciones: desplegar, remover, reposicionar, modificar tamaño
- Restricciones: el radio debe ser positivo



Modularidad

- **"La acción de particionar un programa en componentes individuales puede reducir su complejidad en algún grado...".[Myers]**
- **"La modularización consiste en el hecho de dividir un programa en módulos los cuales puedan ser compilados separadamente, pero éstos tienen conexiones con otros módulos". [Liskov]**
- **"Las conexiones entre los módulos son las asunciones que los módulos hacen acerca de cada otro". [Parnas]**
- **" Modularidad es la propiedad de un sistema que ha sido descompuesto en un conjunto de módulos acoplados coherentemente y libremente". [Booch]**
- **"La meta global de la descomposición en módulos es la reducción del costo del *software* al permitir que los módulos sean diseñados y revisados independientemente... La estructura de un módulo debe ser lo suficientemente simple de forma tal que ésta pueda ser comprendida completamente; debe ser posible cambiar la implementación de unos módulos sin la necesidad de conocer de la implementación de otros módulos y sin afectar su comportamiento".**

Modularidad

- Principio de programación que nos permite precisar las interfaces con distintas decisiones de diseño.
- Los módulos son una forma de organizar y especializar el trabajo de programación.
- Deben ir en un mismo módulo todas aquellas herramientas que tengan una unidad conceptual y de vocabulario, aquellas que se refieran al tratamiento de un mismo aspecto.
- En la medida en que cada uno de estos aspectos este bien distinguido y generalizado en un módulo mayor será la utilidad y rango de aplicación de éste.
- Esta forma de pensar nos permite hacer diseños más amplios y especializados que rebasan las necesidades de nuestra aplicación y nos permiten obtener herramientas más duraderas.
- Se debe buscar completitud.

Encapsulamiento

- **La abstracción y el encapsulamiento son conceptos complementarios: la abstracción focaliza sobre la visión externa de un objeto y el encapsulamiento- también conocido como ocultamiento de información - impide que los clientes vean la visión interna del objeto, donde el comportamiento de la abstracción es implementado. De esta manera, el encapsulamiento proporciona barreras explícitas entre diferentes abstracciones.**
- **"Para trabajar la abstracción, la implementación debe ser encapsulada". [Liskov] En la práctica, esto significa que cada clase debe tener dos partes: una interfaz y una implementación. La interfaz de una clase captura sólo su visión externa, encerrando nuestra abstracción del comportamiento común de todas las *instancias* de la clase. La implementación de una clase comprende la representación de la abstracción así como los mecanismos que alcanzan el comportamiento deseado.**

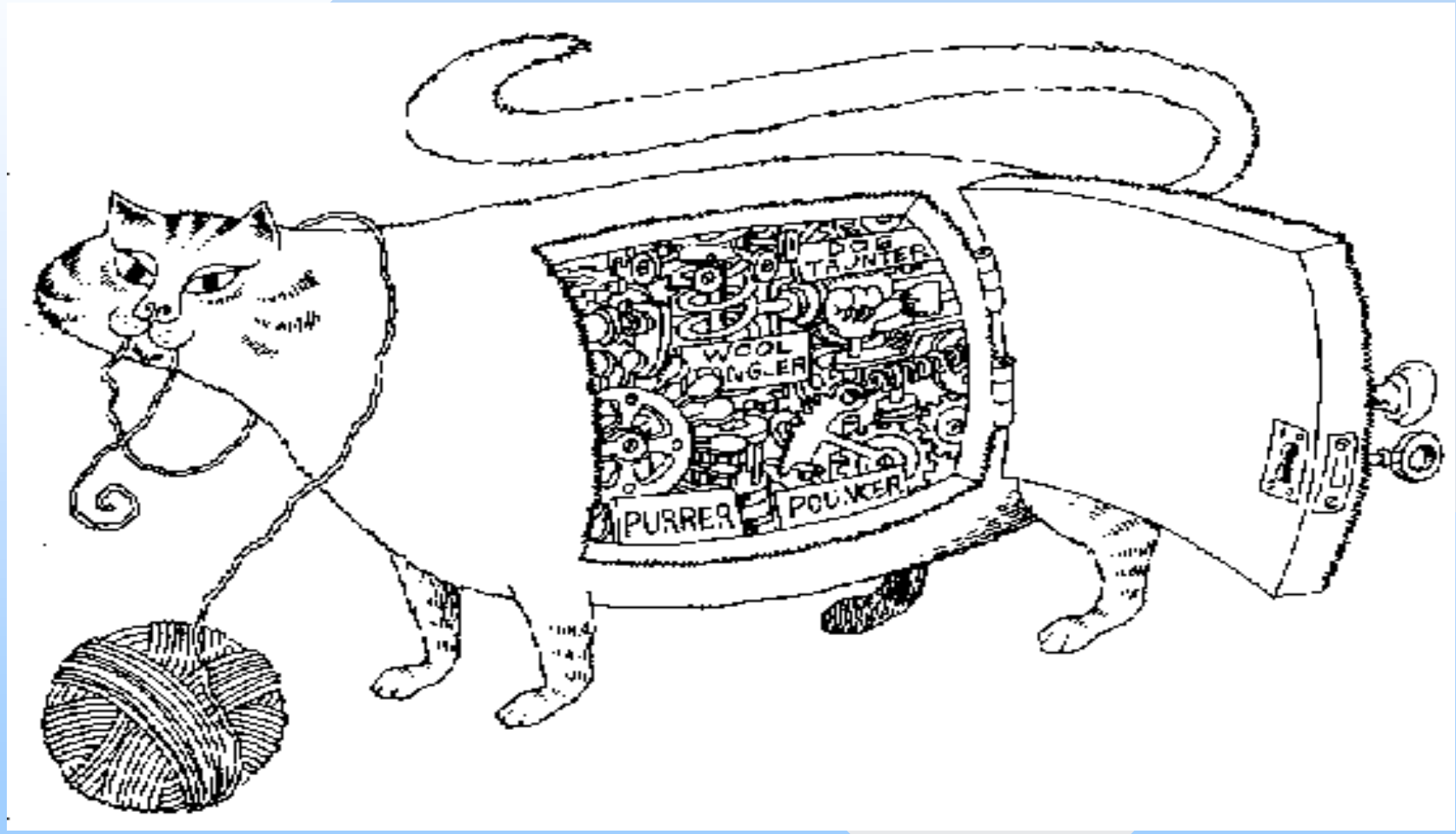
Encapsulamiento

- **"El encapsulamiento es el proceso de ocultamiento de todos los detalles de un objeto que no contribuyen a sus características esenciales". [Booch] En la práctica, uno oculta la representación de un objeto así como la implementación de sus métodos.**
- **"El encapsulamiento (ocultamiento de información) es un principio usado cuando se desarrolla la estructura global de un programa, donde cada componente de un programa debe encapsular u ocultar una única decisión de diseño...La interfaz de cada módulo es definida en forma tal que revele tan poco como sea posible acerca de su funcionamiento interno. [Oxford University]**
- **Esto ayuda a un trabajo eficiente de los programadores de manera que tengan que invertir pocos esfuerzos a la hora de hacer cambios o modificaciones.**

Encapsulamiento (Resumen)

- La abstracción y el encapsulamiento son conceptos complementarios
  - La abstracción se centra en el comportamiento observable de un objeto
  - El encapsulamiento se centra en el origen de dicho comportamiento
- El encapsulamiento
  - Se logra a través del ocultamiento de la información que no contribuye a sus características esenciales de un objeto

Ejemplo de encapsulamiento



Permisos de Acceso

- **Publico:** Los atributos y operaciones declarados públicos (interfaz de la clase) serán del acceso de cualquier función, ya sea ésta una operación en una clase derivada o en cualquier otra clase cliente de los servicios de la clase.
- **Protegido:** Los atributos y operaciones declarados protegidos (implementación de la clase) serán del acceso de cualquier función que sea una operación en una clase derivada.
- **Privado:** Los atributos y operaciones declarados privados (implementación de la clase) serán del acceso sólo de las funciones de la propia clase.

Permisos de Acceso

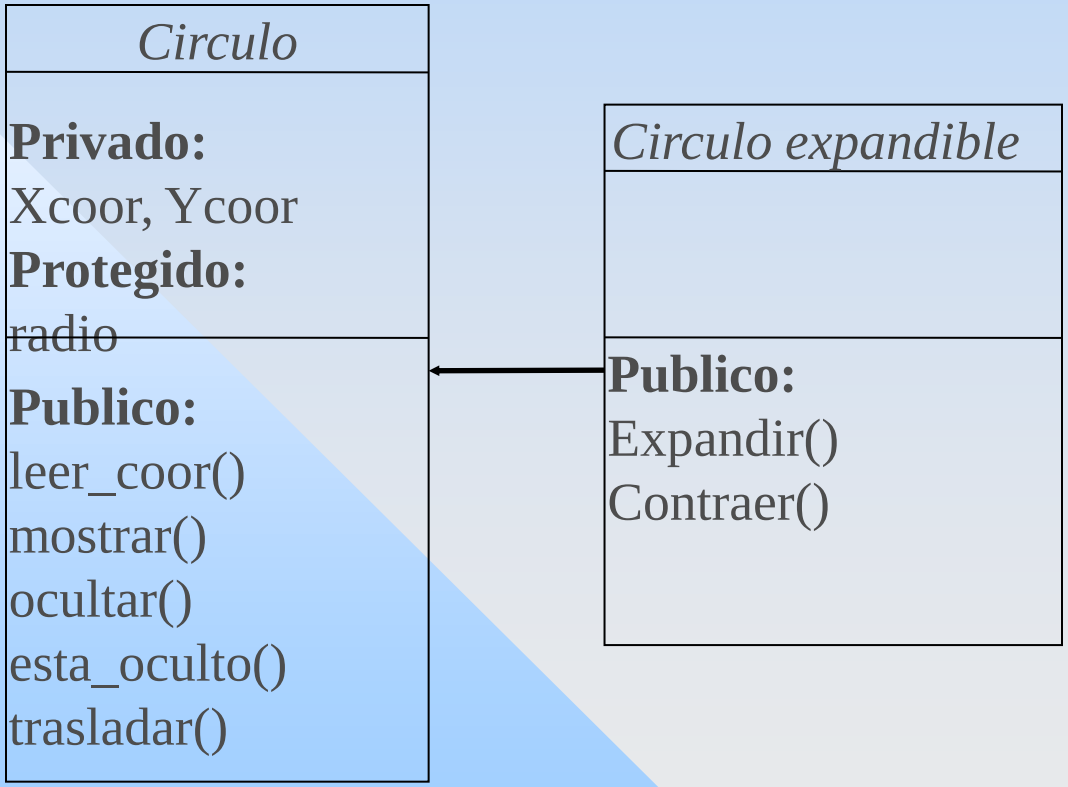
- Tipos de Clientes
  - Operaciones en otras clases que invocan operaciones sobre los objetos de la clase (“simples clientes”)
    - mantener el encapsulamiento
    - Los usuarios no deben tener otra visión de la clase que no sea la que puede tener a través de su interface
  - Operaciones en clases derivadas (“clientes privilegiados”)
    - Tener ciertos derechos a conocer más a fondo los detalles de implementación

Permisos de Acceso
Rompimiento del Encapsulamiento

- Funcionamiento adecuado → Permisos de Acceso
- Sobre
  - Atributos privadas
  - Operaciones privadas

  De de la clase base
- Causas
  - Importancia de la Redefinición
  - Eficiencia

Ejemplo


Circulo
Privado:
Xcoor, Ycoor
Protegido:
radio
Publico:
leer_coor()
mostrar()
ocultar()
esta_oculto()
trasladar()
Circulo expandible
Publico:
Expandir()
Contraer()

Clases y funciones amigas

- En ocasiones un concepto de diseño (una funcionalidad) puede ser vista como la interacción o el trabajo coordinado de objetos pertenecientes a más de una clase.
- En este marco podría ser necesario por ejemplo que las operaciones de cada una de las clases tuvieran accesos plenos a todos los atributos (a toda la implementación) de todas las clases involucradas .

Relación

- Conexión entre cosas
- Utilizadas en el modelado OO
- Se trazan como rutas con diferentes tipos de línea para distinguir diferentes clases de relaciones.

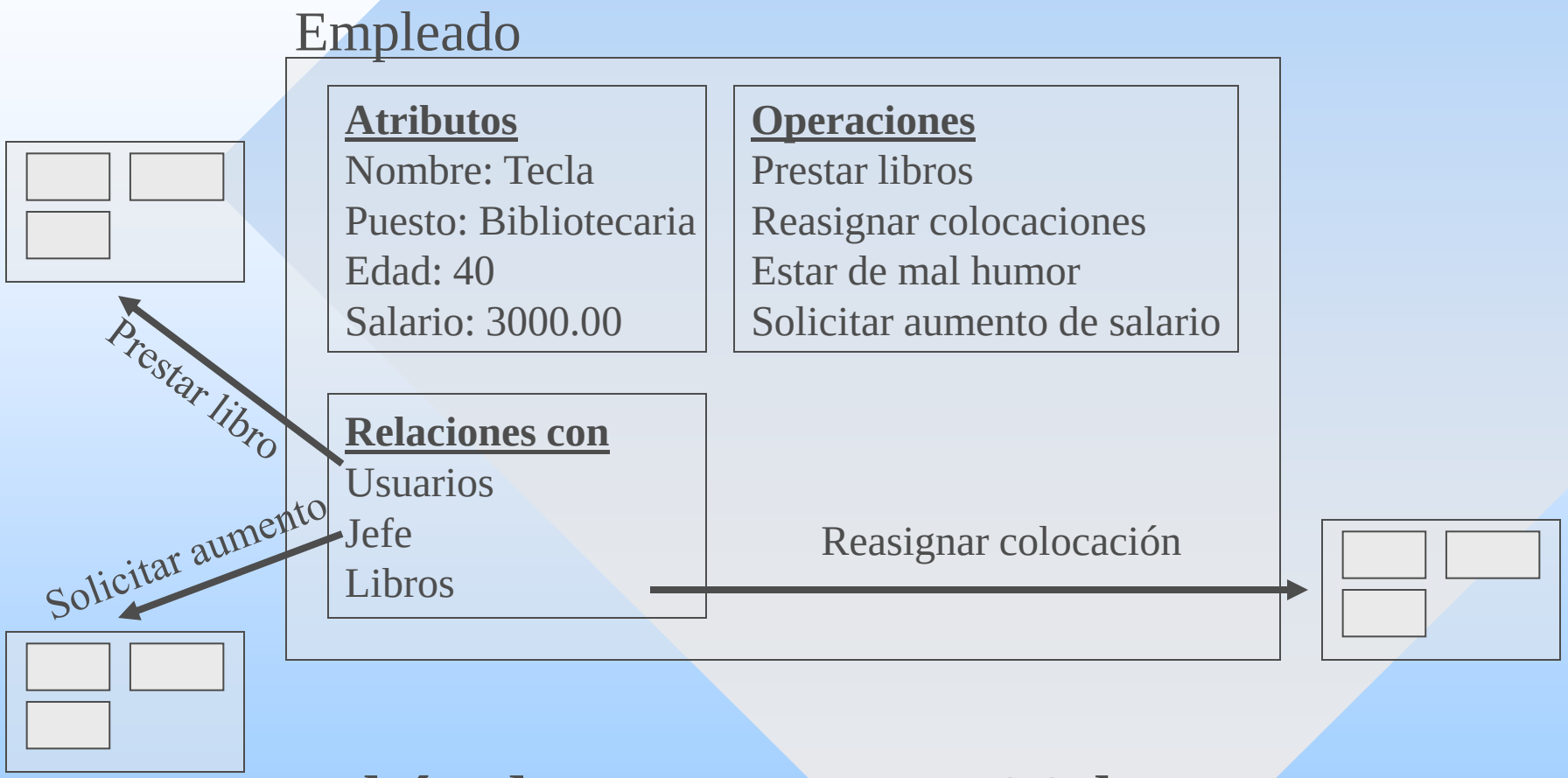
Tipo de Relación: **DEPENDENCIA**

- Es una relación que establece que un cambio en la especificación de un objeto puede afectar otro que lo utiliza, pero no necesariamente a la inversa.
- También conocidas como
  - Relación de **"referencia"**
  - **"asociación de referencia"**

Dependencia

- La conexión sólo se refiere a otro objeto
- La conexión se lleva a cabo a través del paso de mensajes
- Un mensaje consiste de 3 partes
  - Un objeto receptor
  - Una operación que el receptor sabe como ejecutar
  - Un conjunto ordenado de parámetros que esta operación requiere para llevar a cabo su función
    - Parte opcional; si la operación no necesita información adicional para hacer su trabajo, no existen parámetros

Dependencia: ejemplos



Vehículo avanzar: 100 km

Receptor **Operación** **Parámetro**

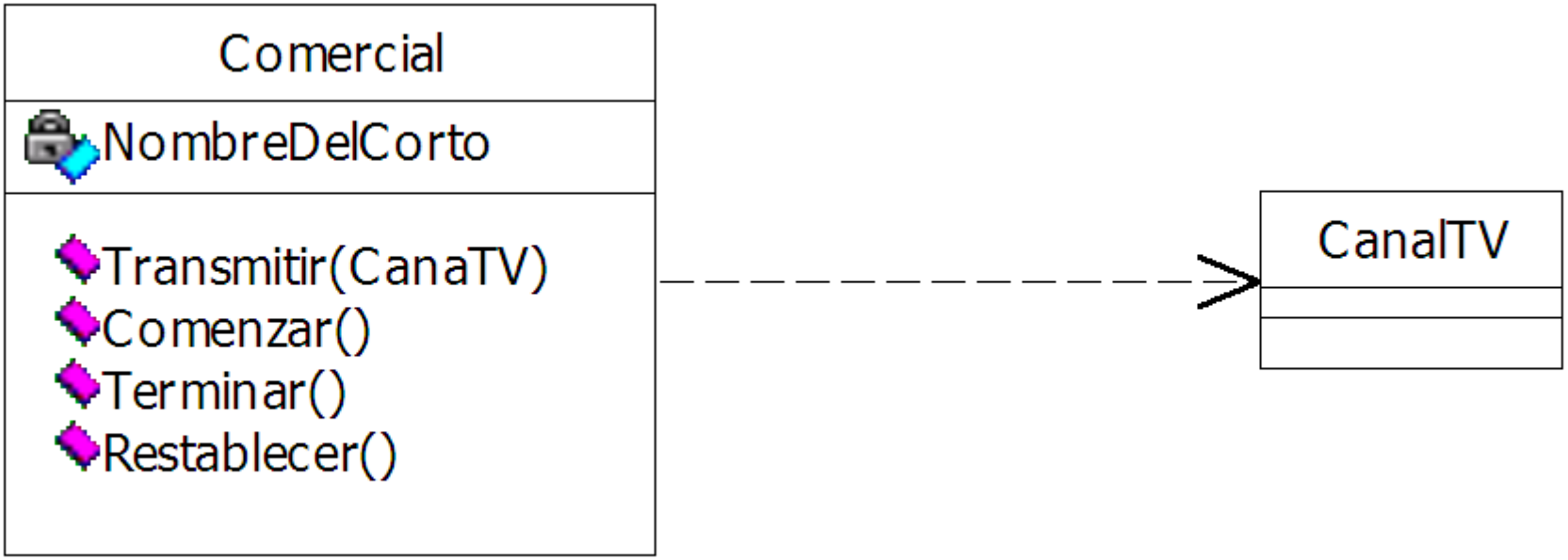
Dependiencia y dinámica de los objetos

- La dinámica de los objetos se genera por los mensajes existentes entre los objetos
- El recibir un mensaje provoca que una operación sea ejecutada por el objeto receptor
- La operación puede mandar mensajes_adicionales a otros objetos

Dependencia: Modelado

- Se traza como una línea punteada dirigida.
- Utilice las dependencias cuando se desea mostrar que un objeto usa a otro.
- Frecuentemente se utilizará cuando una clase utliza a otro como argumento de la firma de una operación.
- La dependencia puede tener nombre cuando se requiera calidad por la cantidad de referencias.

Ejemplo de Dependiencia



Estereotipos de Dependencia

- bind
  - El origen instancía la plantilla destino utilizando los parámetros actuales
- derive
  - El orrigen puede ser calculado a partir del destino
- friend
  - Especifica que el or´gien se le otroga una visibilidad especial dentro del destino
- instanceOf
  - Especifica que el objeto orgien es una instancia del clasficador destino

Estereotipos de Dependencia

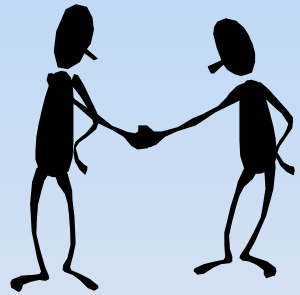
- instantiate
  - El origen crea instancias del destino
- powertype
  - El destino es powertype del origen
  - Powertype es un clasificador cuyos objetos son hijos de un padre dado.
- refine
  - Especifica que el orgien es un grado más fino de abstracción que el destino

Tipo de Relación: **GENERALIZACIÓN**

- Es una relación entre una cosa general (superclase o padre) y una cosa más específica (subclase o hijo).
- "es-un-tipo-de"
- Los hijos pueden ser utilizados dondequiera que el padre aparece.
- Herencia
- Polimorfismo
- Una clase sin padre se llama Raíz (root)
- Tipos: Herencia Simple y Múltiple

Tipo de Relación: **ASOCIACIÓN**

- Relación estructural que especifica que objetos de una cosa son conectos a objetos de otra.
- Son ligas, conexiones o relaciones entre un objeto y uno o más objetos
- Existen asociaciones unidireccionales y bidireccionales, así como estáticas y dinámicas
  - Relaciones estáticas: El acoplamiento de los objetos perdura por un periodo largo de tiempo
  - Relaciones dinámicas: Son aquellas establecidas por las operaciones



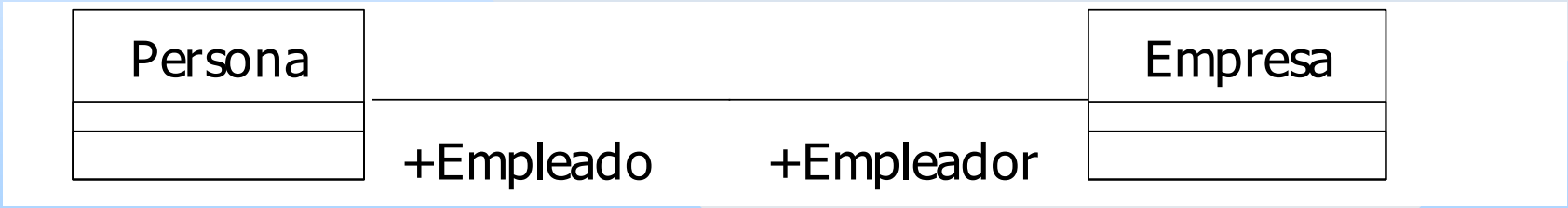
- Al conectarse 2 clases, se puede navegar bidireccionalmente. Existe recursividad.
- Por el # de clases que conectan
  - Binarias
  - n-Elementos
- Adornments
  - Nombre
  - Rol
  - Multiplicidad
  - Agregación

Asociación: Nombre

- Describe la naturaleza y dirección de la relación

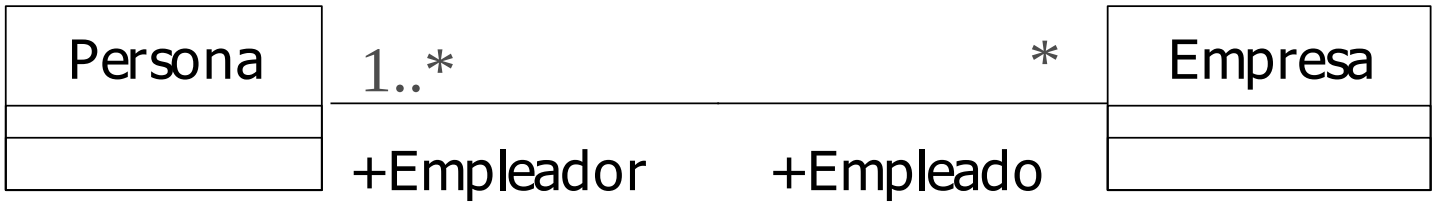
Asociación: Rol

- Faceta de la clase en una relación dada
- Los roles pueden ser explícitos



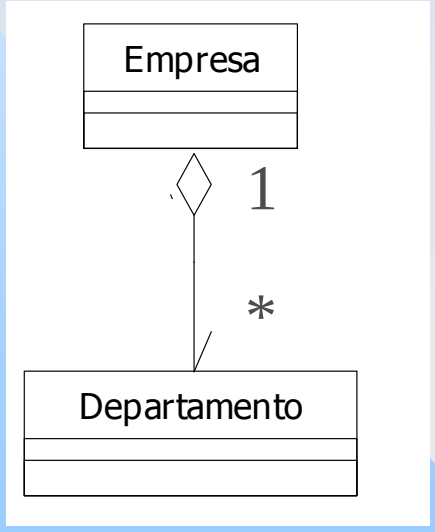
Asociación: Multiplicidad

- Cantidad de objetos a ser conectados a través de una instancia de la asociación



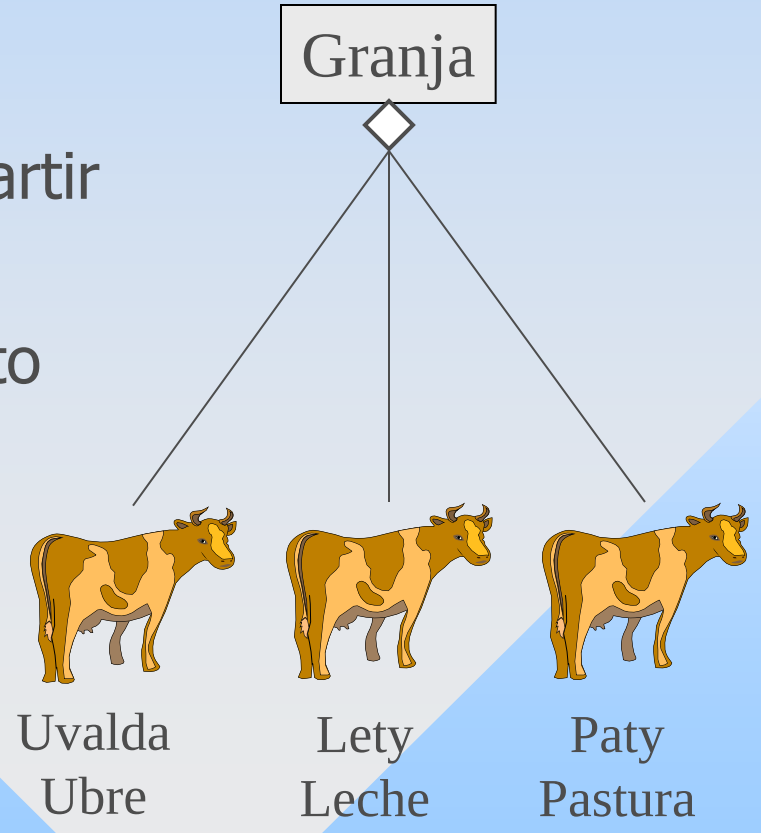
Asociación: Agregación

- Relación estructural entre puntos
- Las dos clases están al mismo nivel, no hay una clase más importante que otra.
- "Todo/Partes" o "Contenedor/Contenido"
- "Tiene"



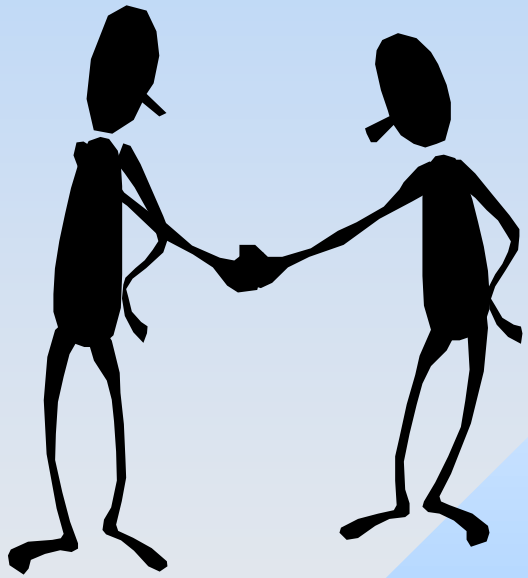
Asociación: Agregación

- Agregación
  - Crea objetos compuestos a partir de objetos simples
  - Es la composición de un objeto como un conjunto de partes



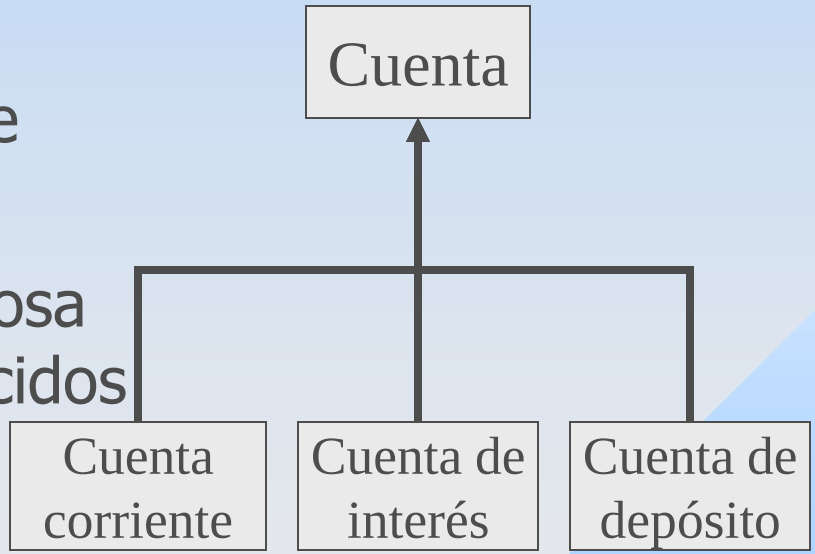
Tipos de asociaciones de agregación

- Existen dos tipos de agregación
  - Normal: Es una relación 'tiene'
    - Una universidad *tiene* facultades
    - Un CD *tiene* pistas
    - Nota: Puede faltar alguna de las partes sin afectar el todo (las partes forman el todo)
  - Composición: Es aquella que posee (*contiene*) a sus partes (fuerte dependencia de propiedad)
    - Un auto *contiene* motor, llantas, puertas, ...
    - Un avión *contiene* motor, asientos, bodega, baños, ...
    - Nota: Si falta alguna de sus partes se afecta el todo (las partes viven el el todo)



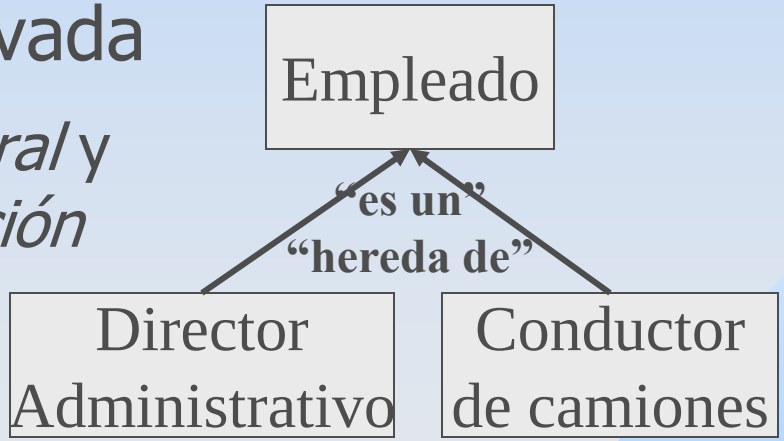
Taxonomía y generalización

- Taxonomía (del griego *taxis*, ordenación + *nomos*, ley)
  - 1 Disciplina que estudia los principios, métodos y fines de clasificación
  - 2 Clasificación de cualquier cosa según unos métodos establecidos
- Una generalización es una relación taxonómica entre un elemento más general y uno más específico



Generalización y especialización

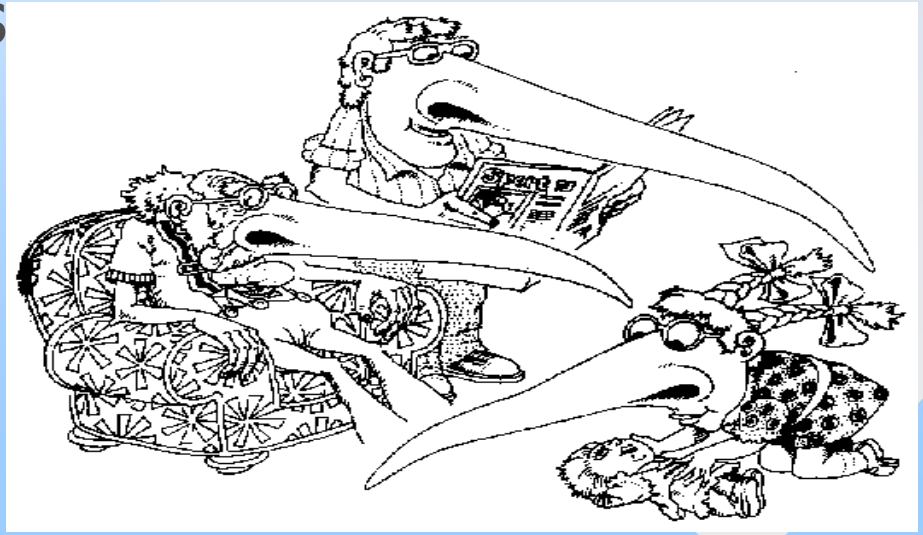
- Generalización implica *generalización-especialización* de la clase base a la clase derivada
  - Donde existe una clase *general* y diferentes clases *especialización*
  - La clase general se llama superclase y las clases especialización se llaman subclases



- Una generalización es una relación 'es un', 'es un tipo de', 'hereda de'

Generalización y herencia

- La generalización cuando se manifiesta en un lenguaje de programación se le conoce como herencia
- La herencia es una relación entre diferentes clases, las cuales comparten características comunes



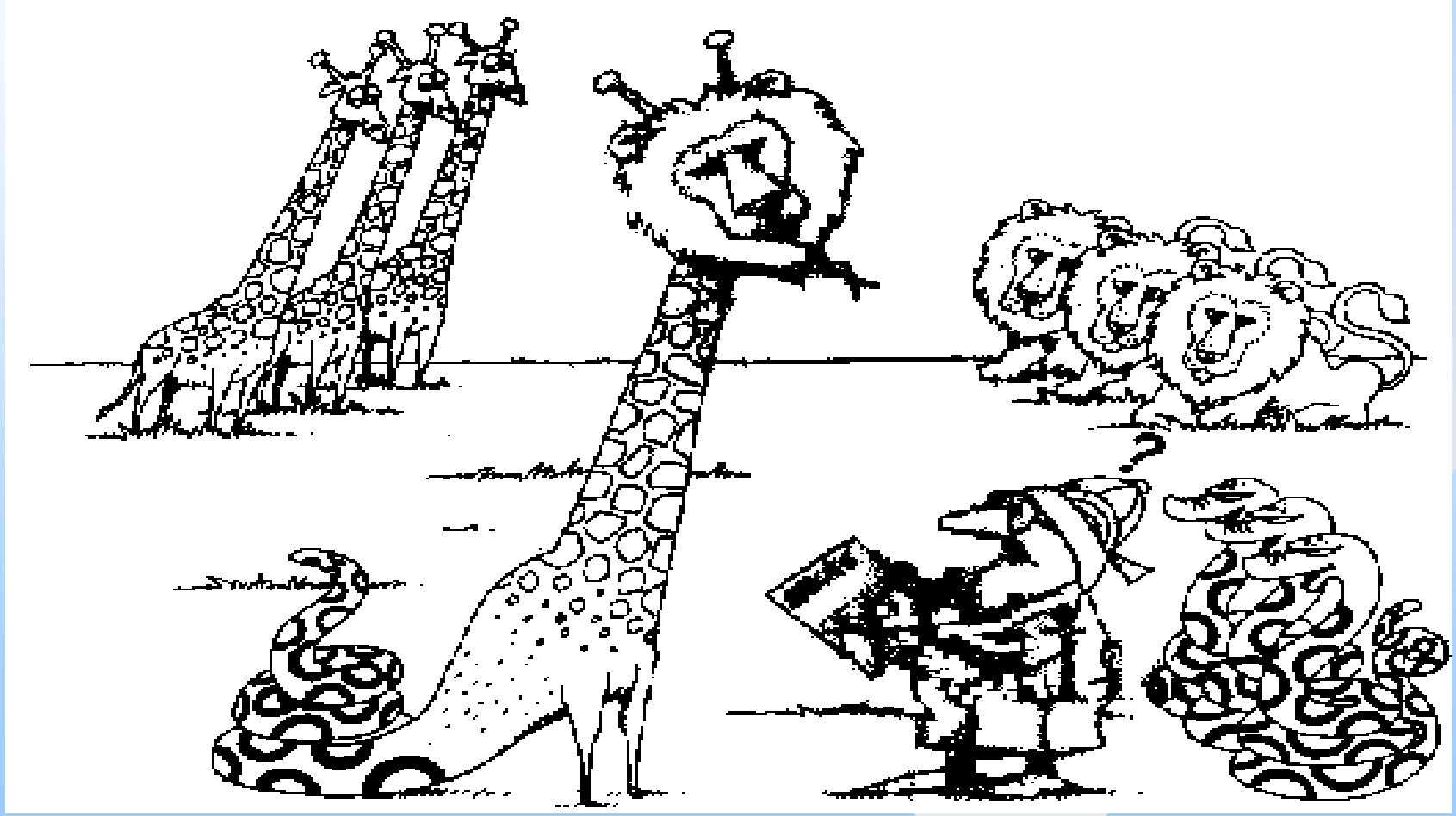
Herencia: Definición

- Es un mecanismo para compartir atributos operaciones comunes entre clases, mediante una relación jerárquica y transitiva, donde una clase hereda de otra sus atributos y operaciones y añade otros nuevos que van a ser exclusivos para ella y sus posibles descendientes, o sea clases que se formen de ella a su vez.

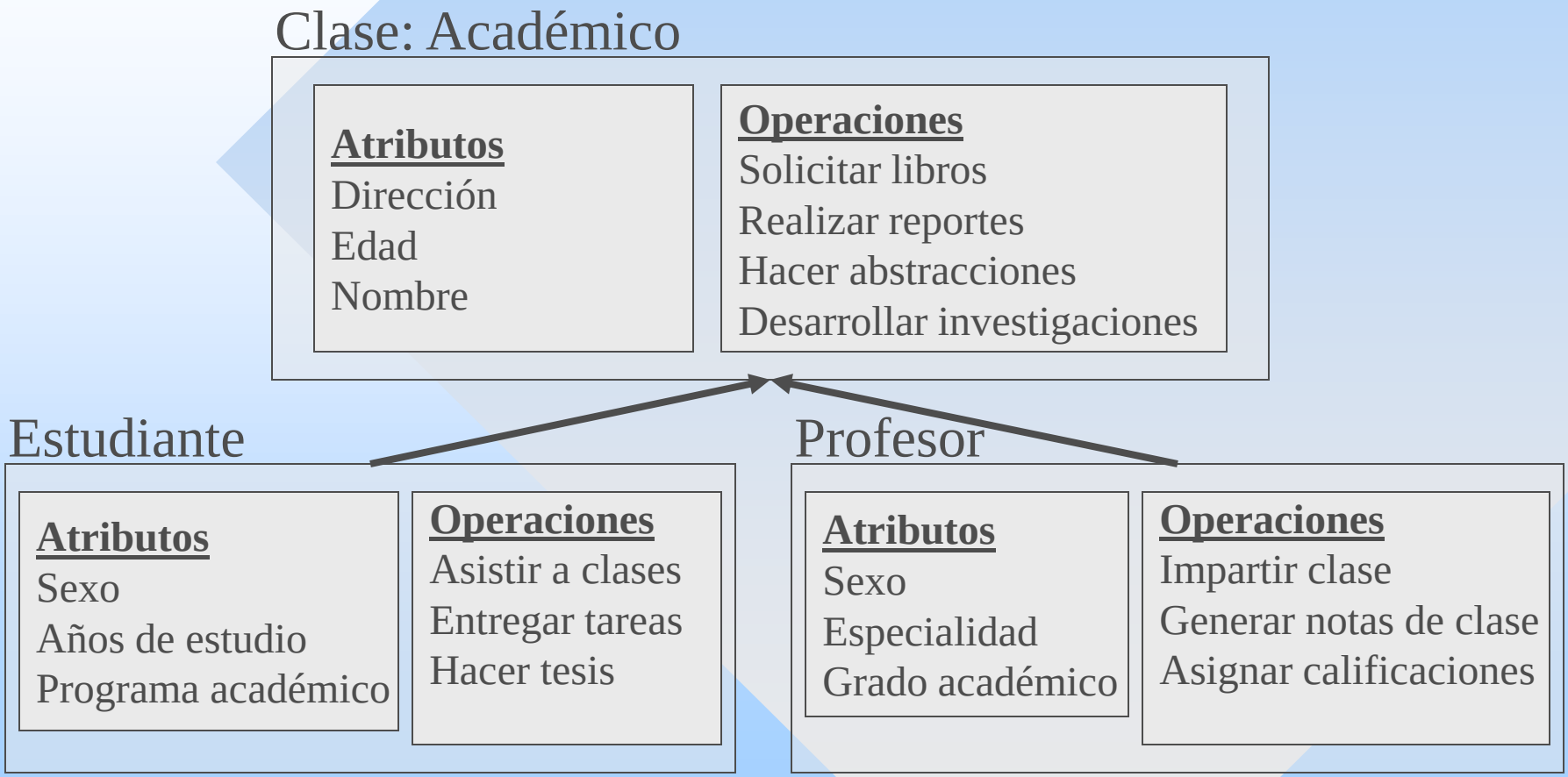
Herencia: Definición

- Es el mecanismo por el cual elementos más específicos incorporan estructura y comportamiento definidos por elementos más generales.
  - UML

Herencia: Ejemplo



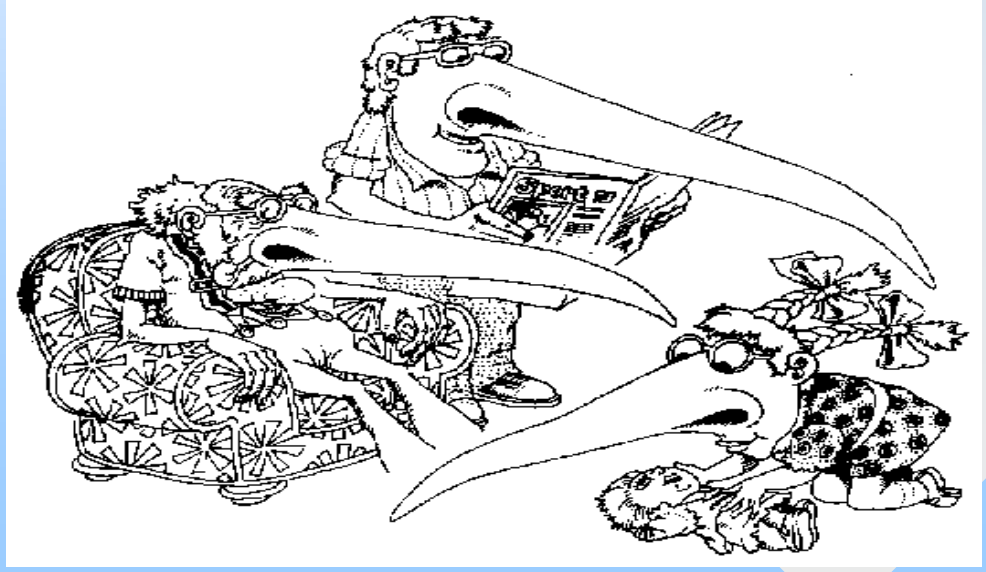
Herencia: Ejemplo



Compartición de atributos y operaciones:
un objeto es la instancia de una clase

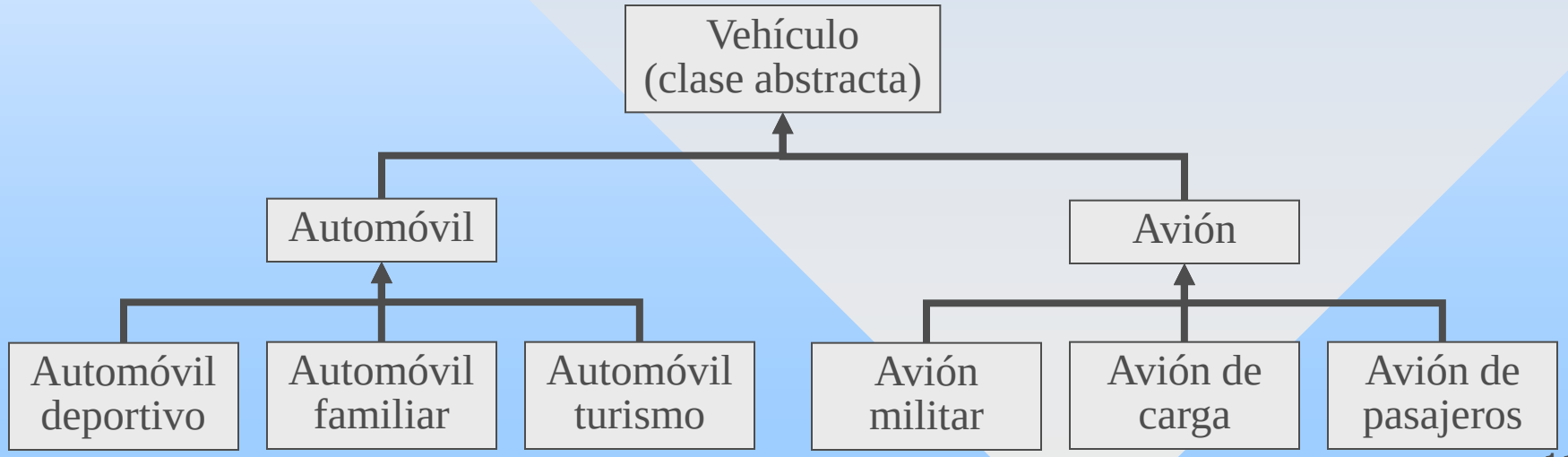
Generalización y herencia

- Una superclase hereda a una subclase y una subclase es heredada por una superclase
  - Se heredan los atributos, las operaciones y todas las asociaciones



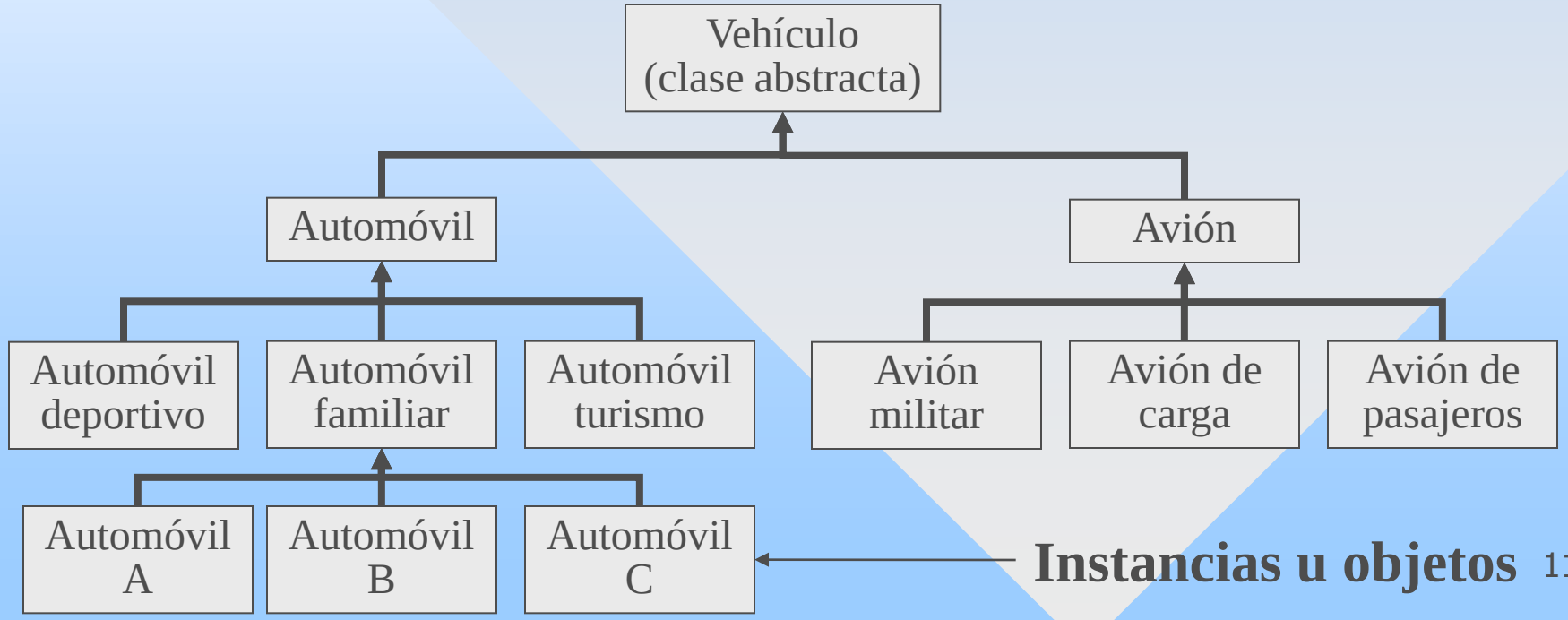
Generalización, herencia y clases abstractas

- Clase abstracta
  - es aquella que no tiene ningún objeto; i.e., no se pueden crear instancias u objetos
  - Sólo se utiliza para heredar de ella y para describir atributos y comportamientos comunes de otras clases



Generalización, herencia y clases concretas

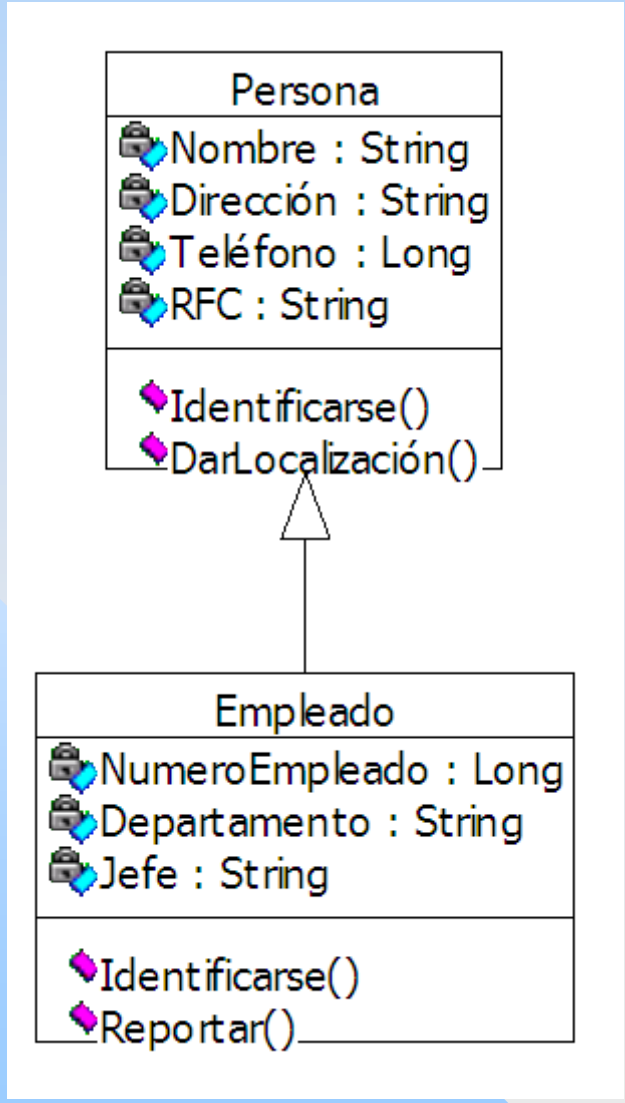
- Clases concretas
  - Son opuestas de las clases abstractas
  - Es posible crear instancias u objetos de ellas y tienen implementaciones de todas las operaciones



Propiedades de la herencia

- Permite reutilización
  - Los componentes pueden reutilizarse
    - Enfoque de reutilización *bottom-up*
  - Los diseños pueden reutilizarse
    - Enfoque de reutilización *top-down*
- Permite claridad en la descripción
  - Debido a que la descripción de objetos en términos jerárquicos es una forma fácil y comprensiva de comunicar tanto estructura como funcionalidad

Herencia: Ejemplo



Polimorfismo

- Una misma operación puede tener comportamientos diferentes según el objeto que la ejecuta.
  - Es decir, el mismo mensaje puede tener respuestas diferentes por parte de objetos diferentes.
- Sin embargo el sentido de la operación sigue siendo el mismo para las dos clases.
  - Luego en este sentido los objetos de ambas clases se dicen polimórficos entre sí.
- Ejemplo
  - CalculaArea(int lado)
  - CalculaArea(int largo, int ancho)

Polimorfismo

- El comportamiento del sistema se define por el comportamiento dinámico de las instancias de las clases
- Según Jacobson
  - Polimorfismo significa que el originador del mensaje no necesita conocer a la instancia de la clase que lo recibe. La instancia receptora puede pertenecer a una clase arbitraria

Polimorfismo

- El polimorfismo permite que diferentes instancias de diferentes clases estén asociadas
- Una instancia receptora interpreta las operaciones de acuerdo a la clase a la que pertenece
- Ejemplo:
  - El mensaje "dibujar" se puede interpretar de diferentes formas por las instancias de una clase

Polimorfismo: Formas de Aplicación

- Se detecta a través de la **"firma"** de los métodos
  - Nombre
  - Número de Argumentos
  - Tipo de Dato de los Argumentos
  - [Valor de Regreso]
- Dentro de la misma clase
  - "Sobrecarga de métodos (funciones)"
  - Caso Especial: Sobrecarga de Operadores
- Entre clases de la misma jerarquía
  - Uso principal del Polimorfismo

Principio de Subclasificación
Liskov

- Este principio establece que toda clase B descendiente de una clase A, no importa a que nivel, puede verse haciendo abstracción como una clase A. O sea, los objetos de la clase B, son también objetos de la clase A, pero no a la inversa. Luego cualquier operación que espere un objeto de la clase A, puede recibir un objeto de la clase B, en tanto los objetos de la clase B saben hacer todas las operaciones que hacen los de la clase A.

Principio de Abierto/Cerrado
Reusabilidad y Extensibilidad

- La máxima utilidad del **polimorfismo** es la redefinición de operaciones de las clases bases en las clases derivadas.
- **Cerrado** = el software ya elaborado no sufrirá modificaciones directas
  - → Reusabilidad
- **Abierto** = se puede definir nuevo software como especialización del anterior
  - → Extensibilidad
- Aplicación de la Herencia

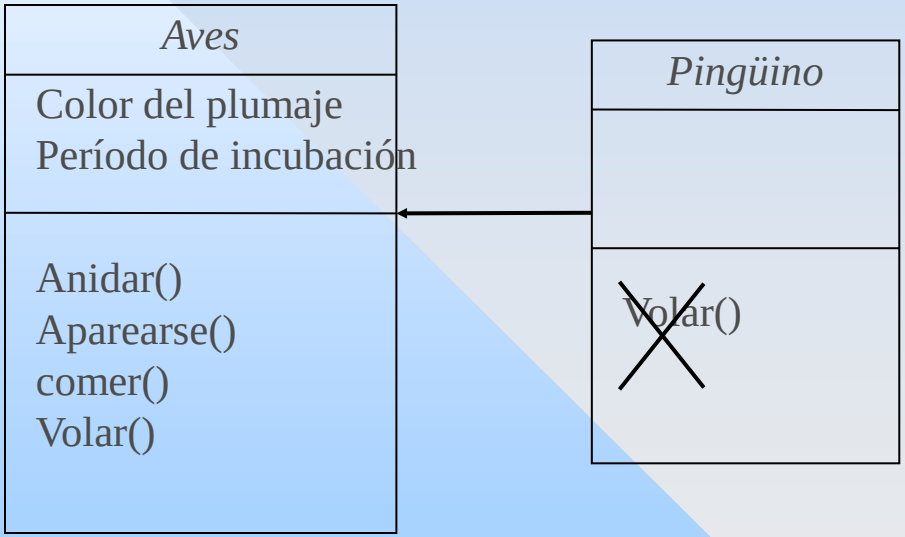
Implicaciones

- La aplicación más importante de la herencia es
  - la simplificación conceptual del problema al reducir el número de características independientes, y
  - mecanismo para formar complejas estructuras de datos por niveles de construcción desde más abstractos hasta más concretos.
- El que una clase comparta atributos y operaciones de otra significa tener en cuenta consideraciones como
  - el encapsulamiento y
  - el mantenimiento de la consistencia de la clase en las clases derivadas.

Problema de Herencia Selectiva

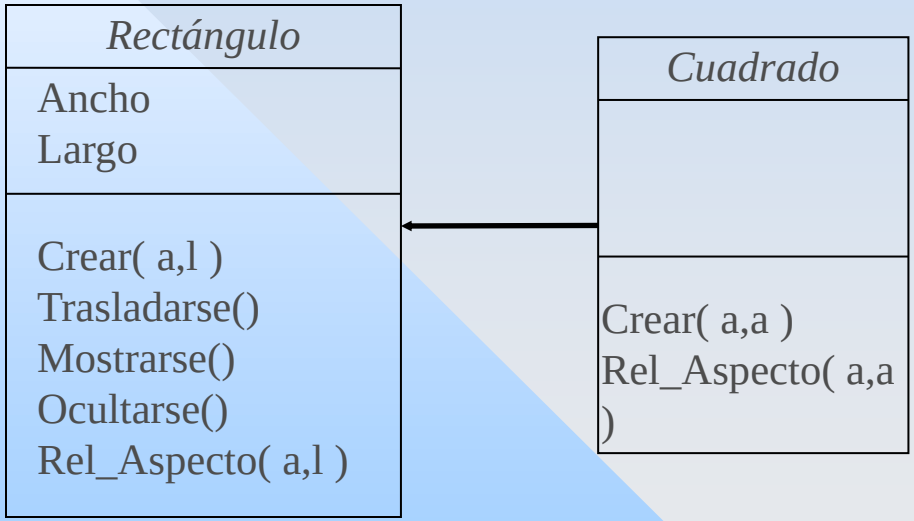
- Anulación de operaciones de la clase base para la clase derivada.
- Aunque un buen diseño de clases exige que se mantenga el principio de subclasificación, o sea, todo lo que tenga sentido para una clase base debe tenerlo también para sus descendientes, en ocasiones se presentan problemas reales donde se precisa de una "herencia selectiva" o "herencia con restricciones".
- En el primer caso se heredará sólo parte del comportamiento de la clase base, que tenga sentido para la nueva clase, es decir sólo una parte de las operaciones. En el segundo, se heredarán todas las operaciones pero algunas de ellas tendrán redefiniciones con restricciones respecto a la forma de sus parámetros.

Ejemplo



Herencia con Anulación

Ejemplo



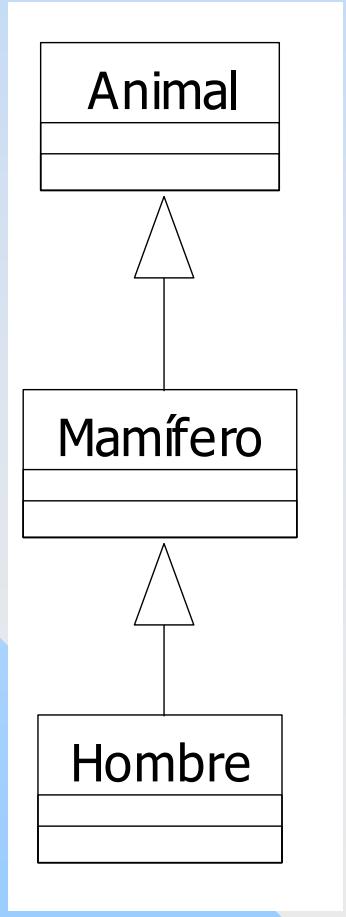
Herencia con Restricciones

Herencia: Consistencia

- La herencia → un mecanismo controlado de redefinición.
- En ocasiones también es necesario contar con herramientas (algunos lenguajes las tienen) para lograr herencias que controlen la redefinición no sólo en interfaz sino también en implementación.
- la clase base se puede adjudicar el derecho de permitir la redefinición de sus operaciones imponiendo ciertas condiciones a la nueva implementación que mantengan la consistencia de la clase. Por ejemplo, que después de la operación se mantenga una cierta relación entre los valores de los atributos.

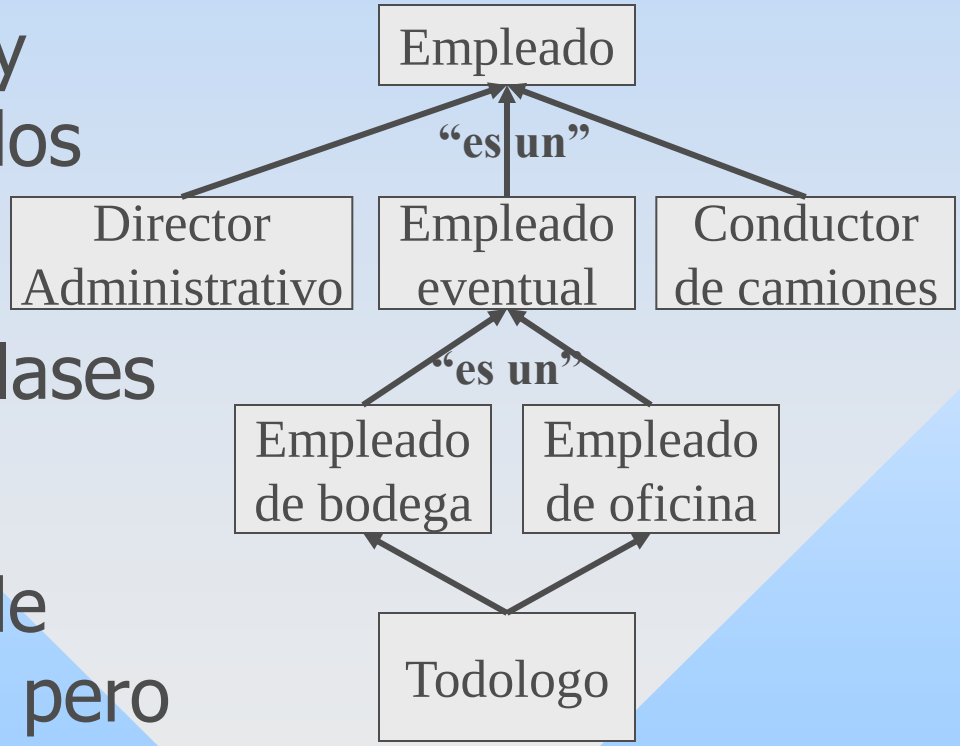
Herencia Simple

- La clase se define o deriva a partir de una única clase base



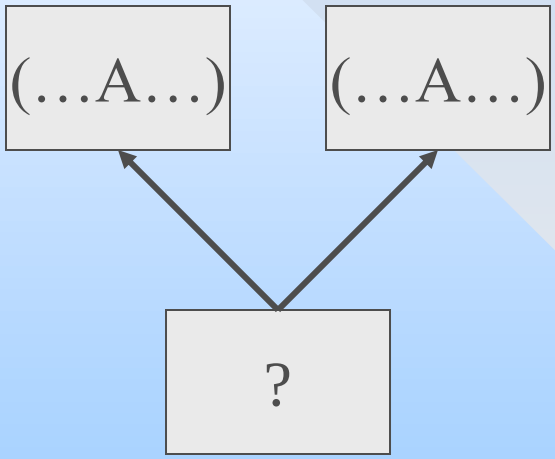
Herencia múltiple

- Es una forma de herencia en donde una clase hereda la estructura y comportamiento de dos clase base
- Existen varias superclases para una subclase
- Permite estructuras de clase más complejas, pero es menos fácil de comprender

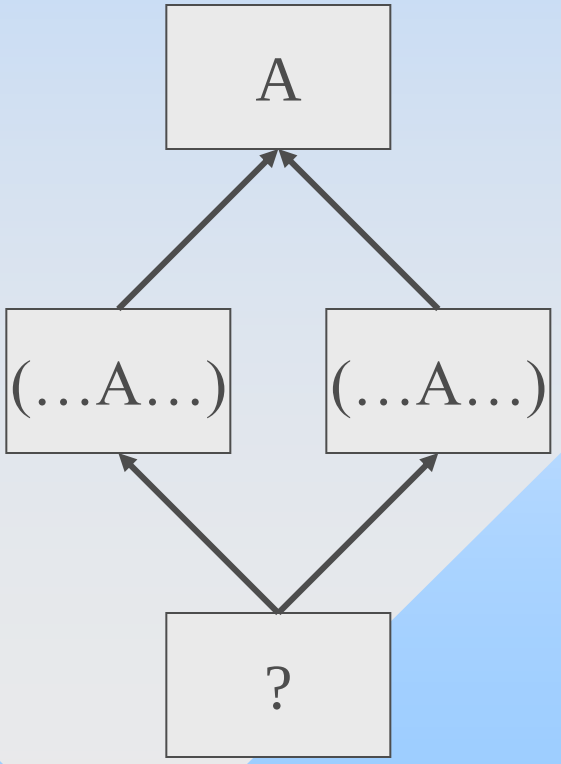


Ambigüedades en la herencia múltiple

Colisión de nombres



Herencia incorrecta repetida



Persistencia

- **Un objeto en software ocupa alguna cantidad de espacio y existe para una cantidad particular de tiempo. "Existe un espectro de existencia de objetos que va desde objetos transitorios que aparecen dentro de la evaluación de una expresión, hasta objetos en una base de datos que sobreviven a la ejecución de un programa". [Atkinson]**
- **Este espectro de persistencia de objetos abarca, entre otras, las siguientes:**
  - **Resultados transientes en la evaluación de una expresión**
  - **Variables locales en activaciones de procedimientos**
  - **Variables propias [como en Algol 60] y variables globales**
  - **Datos que existen entre las ejecuciones de un programa**
  - **Datos que existen entre varias versiones de un programa**
  - **Datos que sobreviven al programa**

Persistencia

- **Los lenguajes tradicionales de programación usualmente direccionan solo los tres primeros tipos de objetos persistentes, la persistencia de los últimos tres tipos es típicamente del dominio de la tecnología de bases de datos.**
- **"La persistencia es la propiedad de un objeto a través de la cual su existencia trasciende en el tiempo (el objeto continúa existiendo después de que su creador deja de existir) y/o en el espacio (la localización del objeto se mueve de la dirección en la cual éste fue creado)". [Booch]**

Conclusiones: Características de la OO

- Reduce la "brecha semántica" entre la realidad y los modelos
- Hace que el sistema sea más fácil de entender
- Permite modificaciones locales de los modelos
- Permite el manejo del cambio
- Promueve la reutilización
- Asegura la continuidad de la representación

Conclusiones: Debilidades de la OO

- Reutilización a gran escala aún no se lleva a cabo
- Existen pocas bibliotecas reutilizables
- La administración de las bibliotecas reutilizables es un problema debido a su falta de estandarización
- Se requiere un entrenamiento fuerte del personal antes de que se pueda implantar

Conclusiones: La OO y los sistemas

- Los sistemas pueden ser
  - Basados en objetos
    - encapsulamiento + identidad de objetos
  - Basados en clases
    - basados en objetos + abstracción de conjunto
  - Orientados a objetos
    - basados en clases + herencia +autorrecursión

INTRODUCCIÓN A UML Y AL ADOO

Alejandro Domínguez

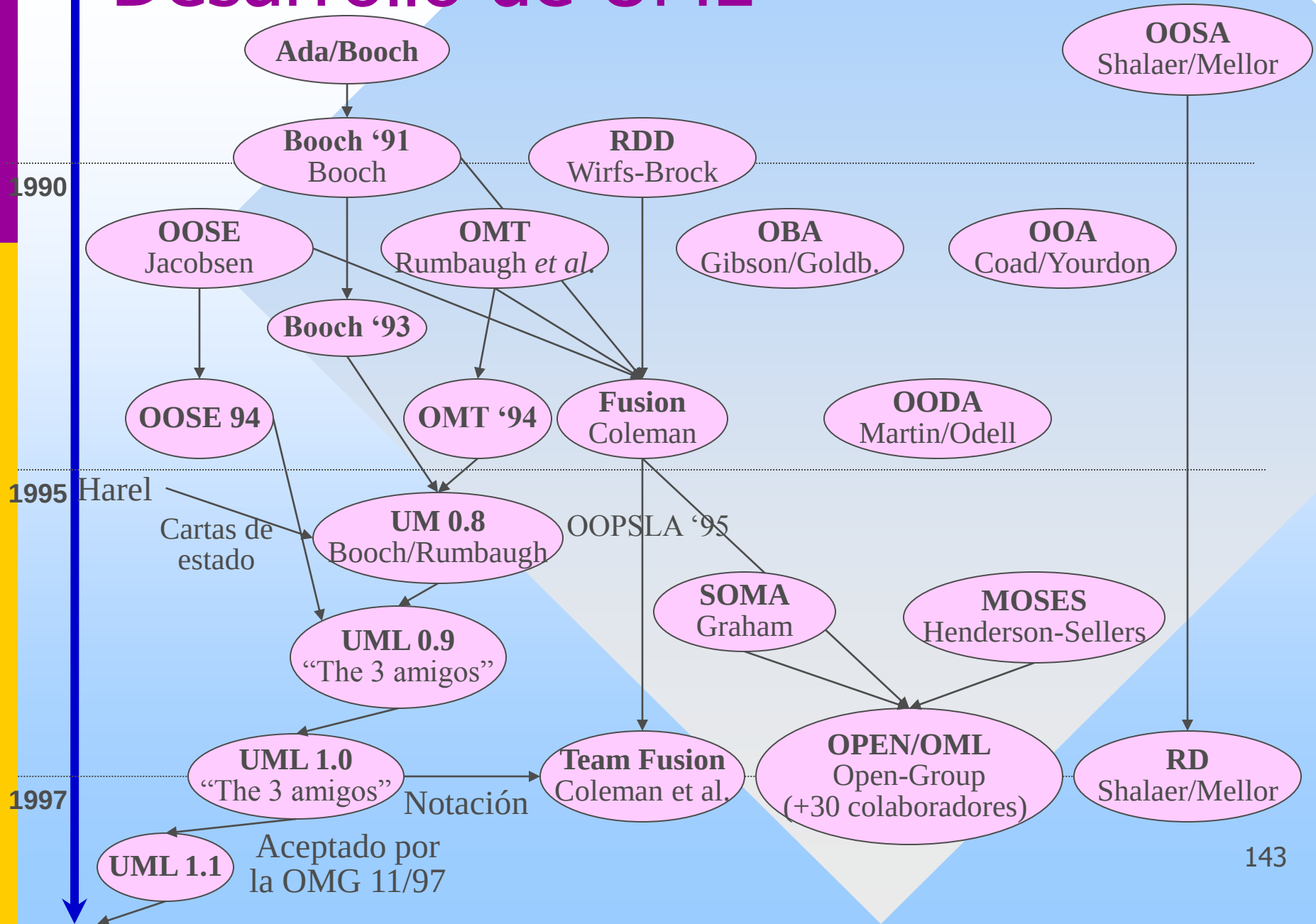
Panorámica general

- Surgimiento y desarrollo de UML
- UML como lenguaje de modelado
- Bloques que componen a UML
- La arquitectura de los sistemas
- Los modelos en UML

Surgimiento de UML

- UML (*Unified Modeling Language:* lenguaje unificado de modelado)
  - Es el sucesor de los diferentes modelos para llevar a cabo análisis y diseño orientados a objetos (ADOO)
  - Nació como una unificación de los modelos de Booch, Rumbaugh (OMT) y Jacobson ("The 3 amigos")
  - Es el lenguaje de modelado estándar aprobado por la OMG en 1997

Desarrollo de UML



UML como lenguaje de modelado

- UML no es un método
- Un método consiste en un lenguaje y en un proceso para modelar
- UML es un lenguaje de modelado
  - Un lenguaje provee un vocabulario y reglas para combinar las palabras en ese vocabulario
  - El modelado conduce a un entendimiento de los sistemas
  - Un lenguaje de modelado es aquel cuyo vocabulario y reglas están enfocadas a la representación conceptual y física de un sistema

Características de UML

- UML es un lenguaje para
  - Visualizar
    - Es un lenguaje gráfico con una semántica bien definida
  - Especificar
    - Permite construir modelos precisos, no ambiguos, y completos
  - Construir
    - UML no es un lenguaje de programación visual, pero los modelos creados con él pueden conectarse directamente a una gran variedad de éstos
  - Documentar
    - Permite expresar, a cualquier nivel de detalle, la arquitectura, los requerimientos, y las pruebas de un sistema

UML y los artefactos del sistema

- UML permite modelar los artefactos inherentes de un sistema
  - requerimientos, arquitectura, diseño, código fuente, planos del proyecto, pruebas, prototipos, versiones, etc.
- Dependiendo de la cultura de desarrollo estos artefactos se tratan a un nivel de detalle diferente

Bloques que componen a UML

- El vocabulario de UML está compuesto por tres grandes bloques
  - Cosas
    - Abstracciones en un modelo
  - Relaciones
    - Unen o ligan las cosas
  - Diagramas
    - Agrupan las colecciones de cosas

El bloque de las cosas en UML

- Existen 4 tipos de cosas en UML:
  - Estructurales
    - Son los sustantivos o nombres de los modelos de UML
      - Casos de uso, clases, interfaces, componentes, nodos
  - De comportamiento
    - Son la parte dinámica de los modelos de UML
      - Interacción entre objetos (mensajes y secuencias de acción) y máquinas de estado
  - De agrupamiento
    - Son las partes organizacionales en los modelos de UML
      - Paquetes (mecanismos de propósito general para organizar elementos en grupos)
  - De notación
    - Son las partes que explican los modelos en UML
      - Notas

El bloque de las relaciones en UML

- Existen 4 tipos de relaciones en UML:
  - Dependencia
    - Es una relación semántica entre dos cosas en la cual un cambio de una cosa afectar la semántica de otra
  - Asociación
    - Es una relación estructural que describe un conjunto de ligas (conexiones entre objetos)
  - Generalización
    - Es una relación de especialización/generalización en la cual los elementos hijo se sustituyen por elementos padre
  - Realización
    - Es una relación semántica entre clasificadores (especifica un contrato que otro clasificador asegura llevar a cabo)

El bloque de los diagramas en UML

- Un diagrama es la representación gráfica de un conjunto de elementos
- UML comprende 8 diagramas:
  - Diagramas de casos de uso
  - Diagramas de clases y objetos
  - Diagramas de secuencia
  - Diagramas de colaboración
  - Diagramas de estado
  - Diagramas de actividad
  - Diagramas de componentes
  - Diagramas de implantación

Arquitectura de los sistemas (1)

- La arquitectura de un sistema se refiere a
  - Su organización
  - La selección de los elementos estructurales, y sus interfaces que lo componen
  - Su comportamiento
    - expresado como colaboraciones entre sus elementos (o subsistemas)

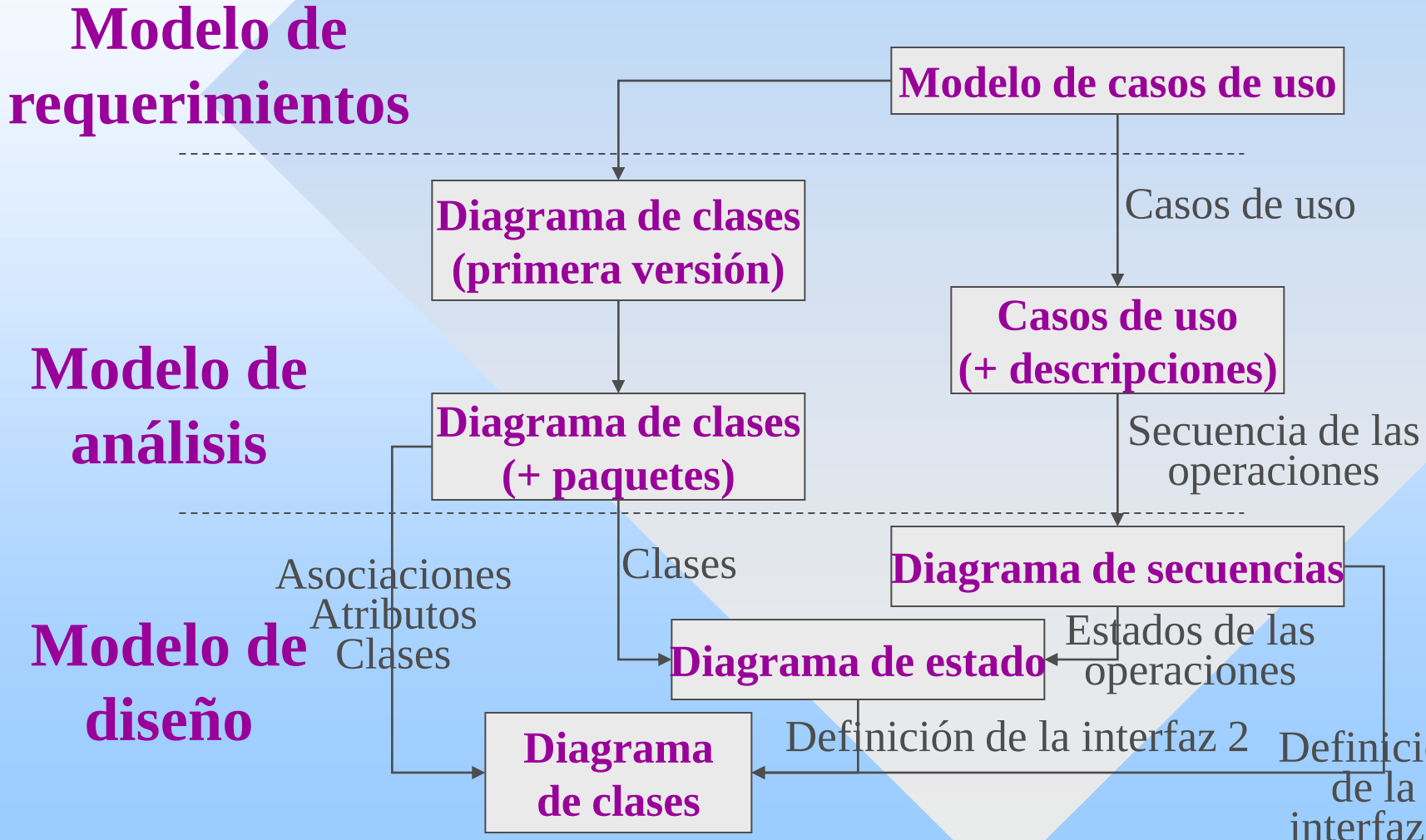
Arquitectura de los sistemas (2)

- La arquitectura de un sistema se refiere a (cont.)
  - La conjunción de estos elementos estructurales y de comportamiento en sistemas mas grandes
  - El estilo arquitectónico que guía esta organización
    - los elementos estáticos y dinámicos y sus interfaces, sus colaboraciones, y su composición

Modelo de una arquitectura de sistemas

La arquitectura como constructora de modelos

Los modelos y la representación en UML: alcance del curso



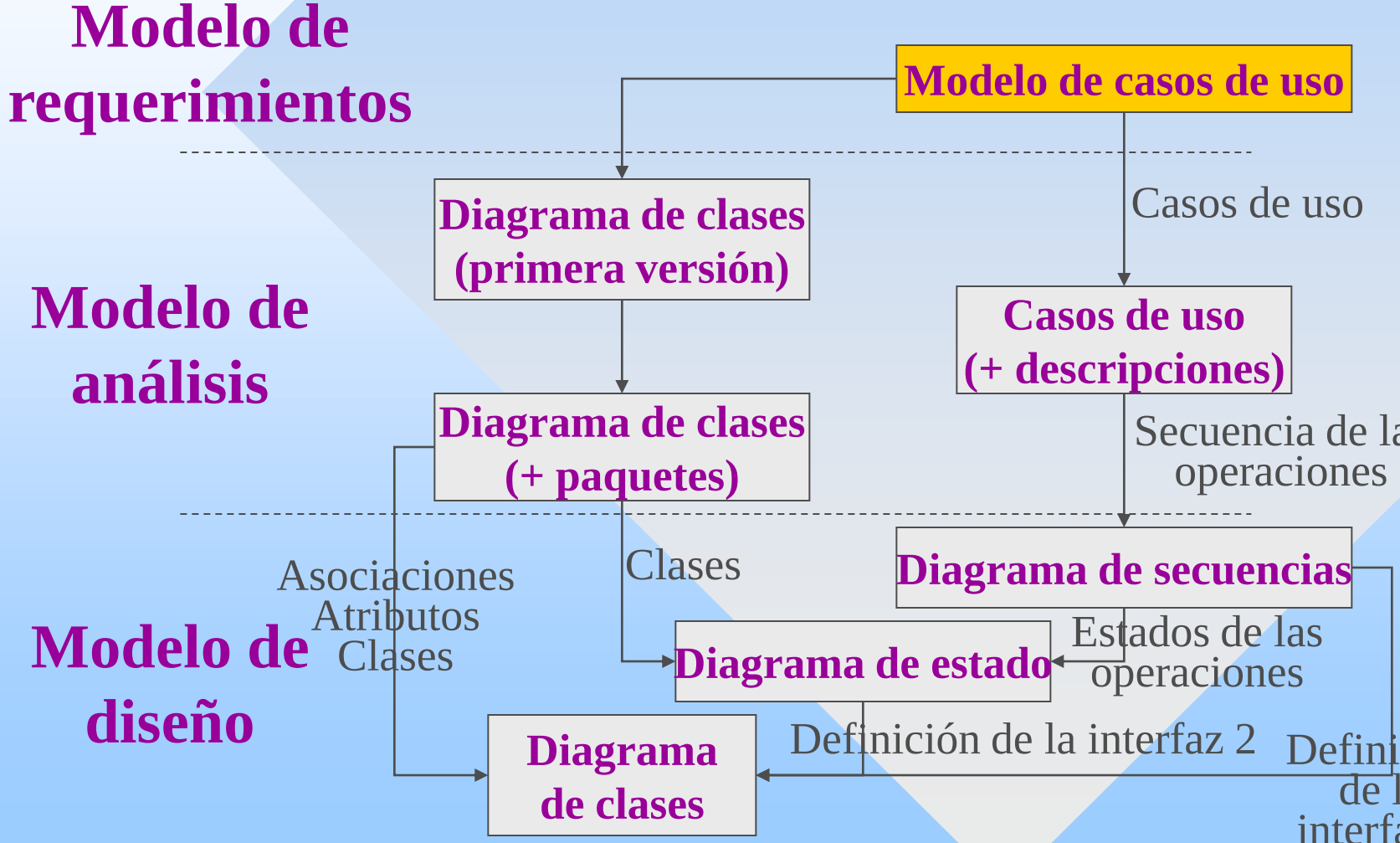
MODELO DE REQUERIMIENTOS (MODELADO DE CASOS DE USO)

Alejandro Domínguez

Panorámica general

- Proporcionar las bases del modelo de casos de uso (*use cases*):
  - Actores
  - Casos de uso
- Notaciones relevantes de UML

Entorno del modelo de casos de uso



Características de los requerimientos de un sistema

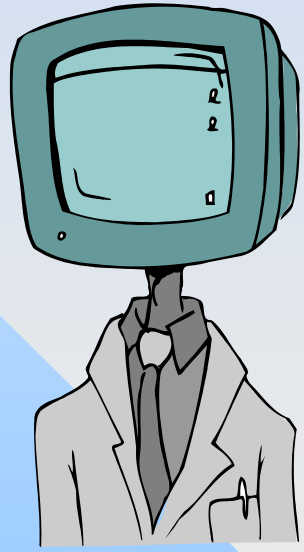
- Los requerimientos de un sistema
  - Necesitan ser los adecuados
  - Necesitan estar más enfocados al usuario
  - Son los primeros en someterse a un proceso de reingeniería
  - Deben involucrar 100% al usuario
- Los requerimientos de un sistema deben centrarse en los usuarios

Análisis centrado en los usuarios

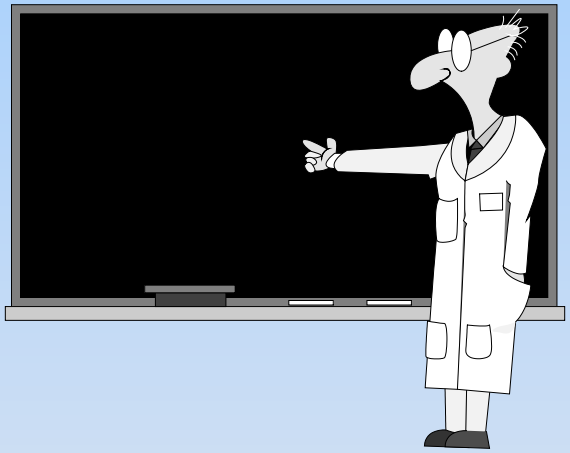
- En un proceso para capturar los requerimientos desde la perspectiva de los usuarios
- Seguido de
  - Análisis
    - Explora la conectividad y las consecuencias de los conflictos (diferentes y potenciales) que surgen en los requerimientos del usuario
  - Diseño
    - Mapea los requerimientos en una aplicación de software

Obtención de los requerimientos del usuario

- El usuario debe decir que desea
- El analista debe
  - Entender el negocio del usuario
  - Conocer las características de todas las cosas y personas que interactuarán con el sistema
  - Tomar tiempo para explorar las necesidades del usuario
  - Observar y tomar nota de cómo el usuario trabaja
  - Estructurar la información de tal forma que pueda ser utilizada más tarde en el análisis y diseño
  - Descubrir, con ayuda de expertos, las reglas de negocio críticas
  - Enfocarse en el *qué* y no en el *cómo*



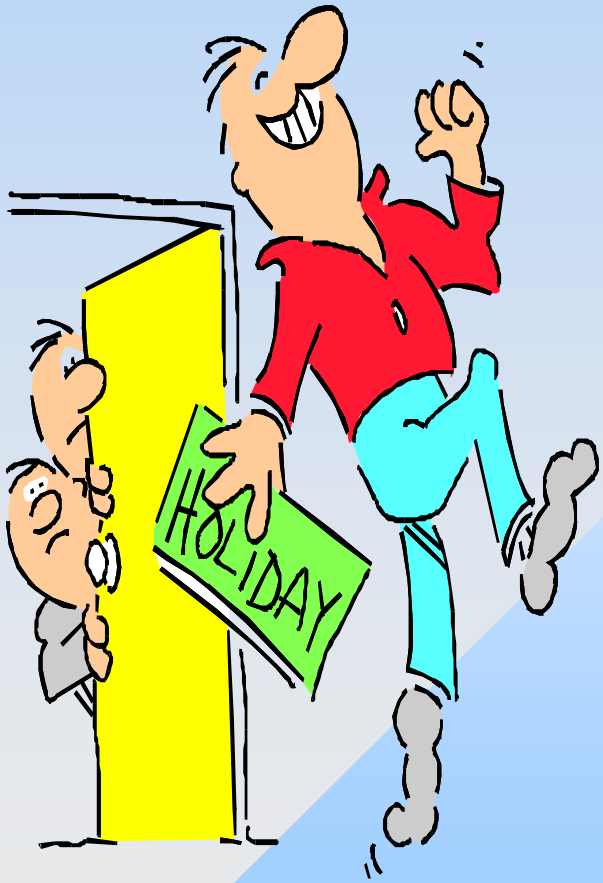
Durante y después de los requerimientos



- El analista debe
  - Escribir la información recolectada de tal forma que el usuario la comprenda
  - Crear una serie de diagramas acerca del negocio del usuario
    - Ayudan a la comprensión
    - Proporcionan un punto de partida para confirmar los requerimientos reales
- Los modelos de casos de uso ayudan llevar a cabo lo anterior

¿Qué es un modelo de casos de uso?

- Un modelo de caso de uso:
  - Describe los servicios que el sistema proporciona al usuario
  - Proporciona información acerca de los usuarios (actores) del sistema
    - Actores: Humanos, otros sistemas, máquinas
  - Muestra la naturaleza de las interacciones entre el actor y el sistema (casos de uso)
    - El sistema contiene los casos de uso
  - Relaciona actores y casos de uso



Producción de un modelo de casos de uso

INPUTS

- Prácticas existentes de sistemas
- Requerimientos del nuevo sistema

PROCESOS

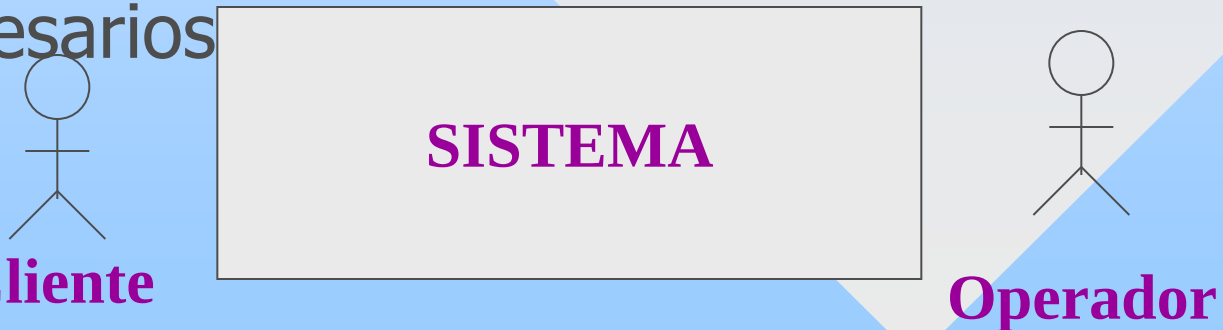
- Derivar posibles actores y casos de uso
- Discriminar sobre posibles caso de uso
- Identificar asociaciones entre casos de uso

OUTPUTS

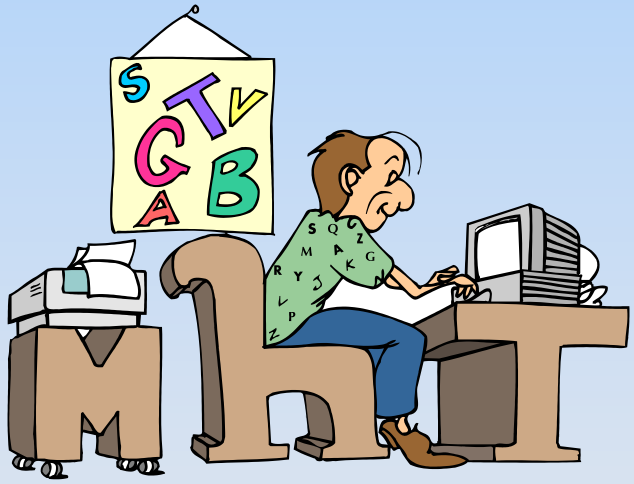
- 1 modelo de casos de uso

Actores

- Un actor
  - Es alguien o algo externo al sistema
    - Personas, software, hardware, almacenes de datos, o redes
  - No sigue acciones precisas
  - Representa cualquier cosa fuera del sistema y que interactuará con él
    - Puede ser humano o un máquina
- Los detalles acerca de los actores son no necesarios



¿Quiénes pueden ser actores?



- Los usuarios del sistema
- Los instaladores del sistema
- Los operadores del sistema
- Los que dan soporte técnico al sistema
- Otras sistemas que utilizan el sistema
- Los que obtienen información del sistema
- Los que proporcionan información al sistema

Actores primarios

- Son aquellos para los cuales se especifica y se construye
- Son activos
- Inician su actividad con el sistema
  - Usuario de computadoras con la computadora
  - Usuario del teléfono con el teléfono
  - Cobrador con el sistema de cobranza
  - Interanuta con el visualizador web
- Obtienen siempre algo a cambio

Actores secundarios

- Supervisan, ayudan y mantienen al sistema
  - Son pasivos: no inician ninguna actividad con el sistema
- Sólo existen para que los actores primarios puedan utilizar el sistema
  - Siempre disponibles cuando el sistema necesita de su ayuda
- Pueden ser máquinas u otros sistemas
  - Impresoras, plotters, modems, ...
  - Aplicaciones adicionales
  - Posiblemente actores humanos

Características de los actores

- Un actor no es un usuario en particular
  - Un actor representa el papel que juega el usuario
  - Un usuario es alguien que juega un papel específico mientras usa el sistema
    - Juan Pérez es un alumno
- Cada actor utiliza el sistema de formas diferentes
  - De otra forma debería ser el mismo actor
- Cada forma en que el usuario utiliza el sistema es un *caso de uso*

Casos de uso (1)

- Describen las transacciones ofrecidas por el sistema
- Son iniciados por los actores
- Pueden ser llamados por otros casos de uso
- Pueden ser combinados para ofrecer mayor funcionalidad

- Representan lo *qué* el sistema debe proporcionar, en lugar de *cómo* lo debe hacer
  - Cómo el sistema proporcionará el servicio no debe ser importante para el usuario
    - Cuando se utiliza el teléfono , no es importante cómo se lleva a cabo la conexión con otro

Casos de uso (2)

- Un **caso de uso**:
  - Es una serie completa de caminos de las acciones (con eventos de inicio y término, con actores involucrados y objetos utilizados en la acción) y las reglas que gobiernan cómo se ligan esas acciones
  - Es la definición de la interacción entre el sistema y el actor
  - Incluye los caminos normales y alternativos
  - Es parecido a un proceso de interacción humano-máquina

Casos de uso (3)

- Los casos de uso existen debido a las necesidades del usuario
  - De esta forma se garantiza que el sistema hará lo que el usuario espera
- Los casos de uso no son escenarios
  - No son recolectores de un conjunto particular de interacciones entre el usuario y el sistema
  - Los casos de uso surgen a partir de los escenarios

Casos de uso y escenarios (1)

- Un escenario
  - Es un orden específico de eventos
  - Es una sesión que un actor tiene con el sistema
  - Tiene detalles de los datos reales y de la salida esperada
  - Puede ser ligeramente diferente que el previo, aún cuando se haga esencialmente lo mismo

Casos de uso y escenarios (2)

- Pueden existir cientos de escenarios en un aplicación
- Los escenarios son importantes fuentes de información para descubrir casos de uso
- Mas de un escenario se puede generar de un simple caso de uso si una o más reglas para ligar a las acciones no están en secuencia estricta

Casos de uso y escenarios (3)

- Los casos de uso proporcionan un conjunto potencial de escenarios
- Observando una familia de escenarios similares, se puede obtener la esencia de lo que se está llevando a cabo
  - Escenarios similares seguirán patrones similares y proporcionarán tipos similares de resultados

Casos de uso y escenarios (4)

Ejemplo de escenario 1

- Juan teclea la cuenta 12345
- Juan teclea su NIP 54321
- Juan su balance promedio entre 1/1/1995 - 31/12/99
- El sistema proporciona el balance promedio

Ejemplo de escenario 2

- María teclea la cuenta 23457
- María teclea su NIP 75432
- María su balance promedio entre 1/1/1995 - 30/12/99
- El sistema proporciona el balance promedio

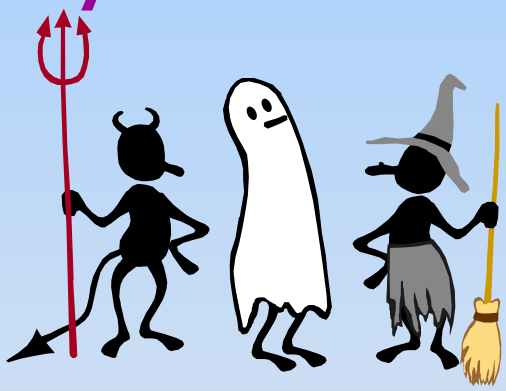
Ejemplo de escenario 3

- Pedro teclea la cuenta 23456
- Pedro teclea su NIP 65432
- Pedro su balance promedio entre30/11/1999 - 30/12/99
- El sistema proporciona el balance promedio

Casos de uso y escenarios (5)

- Existen dos tipos de escenarios
  - Primarios
    - Describen lo que sucede en un flujo normal de interacción con el sistema
    - Existen cuando
      - Todo funciona bien
      - No existen *bugs*
      - No existen errores
  - Secundarios
    - Describen las excepciones encontradas en un flujo normal (flujos alternativos)
    - Existen cuando
      - Una acción debe tomarse en algún punto
      - Algo es erróneo en algún punto
      - Algún comportamiento que puede pasar en algún momento

Descripción de actores y casos de uso



- Cada actor necesita
  - un nombre descriptivo
  - una breve descripción de una o dos oraciones de longitud
    - Las sentencias deben describir al actor con respecto al sistema
- Cada caso de uso necesita
  - Un nombre descriptivo
  - Una descripción de una o dos oraciones de longitud
    - Sólo si el nombre no dice suficiente acerca de la funcionalidad del caso de uso

Ejemplos de descripción de actores

- Actores
  - Cliente - una persona quien ordena productos de la compañía
  - Vendedor - un empleado de la compañía quien procesa los pedidos del cliente
  - Mensajería - UPS, FedEx, DHL, MexPost, etc.
  - Empacador - Un empleado de la compañía que empaca, etiqueta y envía ordenes
  - Sistema de inventario - software que almacena y procesa la información del inventario de productos
  - Sistema contable - software que almacena y procesa la información contable

Práctica

- Tabla de Actores

Nombre del Actor	Tipo **Primario** **Secundario** **Ambos**	Descripción

Ejemplos de descripción de casos de uso

- Casos de uso
  - Del cliente - elaborar orden, obtener estado de orden, regresar producto, cancelar orden, registrar observaciones
  - Del vendedor - elaborar orden, obtener estado de orden, regresar producto, cancelar orden, registrar observaciones
  - A la mensajería - enviar paquetes listos para envío
  - Del empacador - imprimir etiqueta de correo, calcular cuota postal
  - Al inventario del sistema - proporcionar información del producto, actualizar inventario
  - Del inventario del sistema - capturar las ordenes recibidas
  - Al sistema contable - cargar a cuenta, proporcionar crédito

Práctica

- Relación de Actores con Casos de Uso

Actor	Dirección	Casos de Uso

El modelo de casos de uso y la descripción de sistemas (1)

- El modelo de casos de uso caracteriza el comportamiento de todo el sistema y los actores externos
- Un sistema se describe por un modelo de casos de uso
  - El modelo contiene un conjunto finito de casos de uso

El modelo de casos de uso y la descripción de sistemas (2)

- Un sistema potencialmente tiene un número infinito de escenarios
  - Familias de escenarios generan casos de uso y éstos a su vez el modelo de casos de uso
- Cada caso de uso en el modelo debe ser numerado, de otra forma el sistema no será funcionalmente completo

El modelo de casos de uso y la descripción de sistemas (3)

- Con el modelo de casos de uso el sistema puede ser puesto a prueba en sus entradas (*inputs*), salidas (*outputs*) y naturaleza de la interacción

Notación gráfica en un diagrama de casos de uso

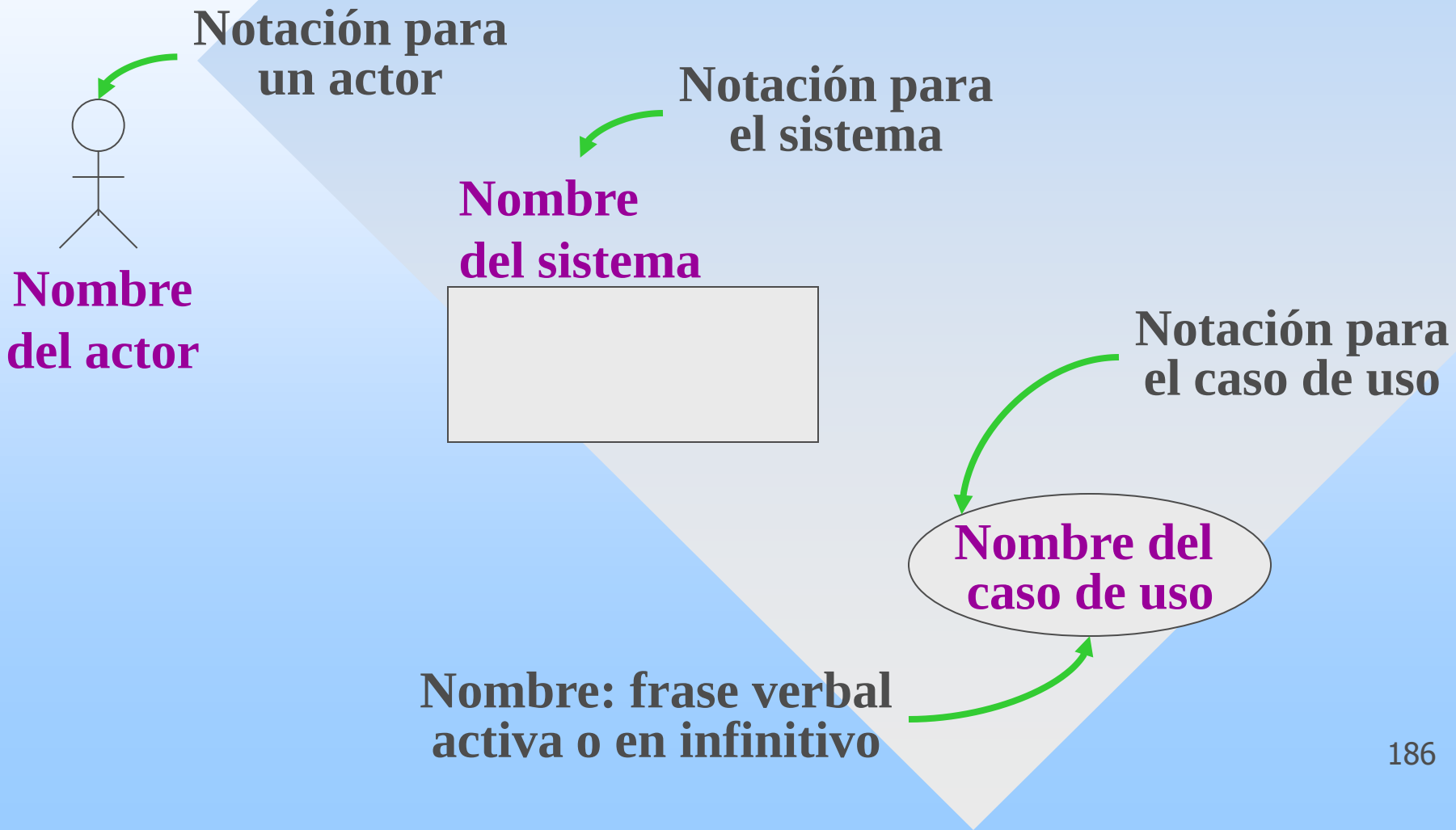
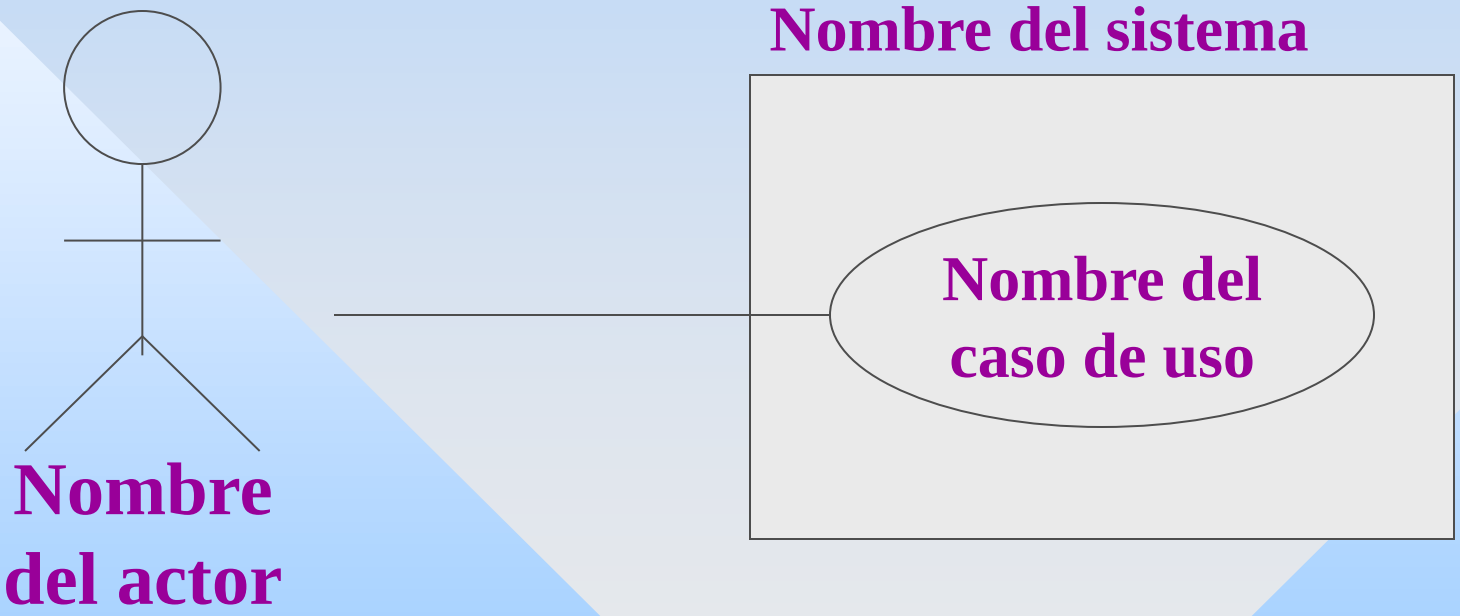
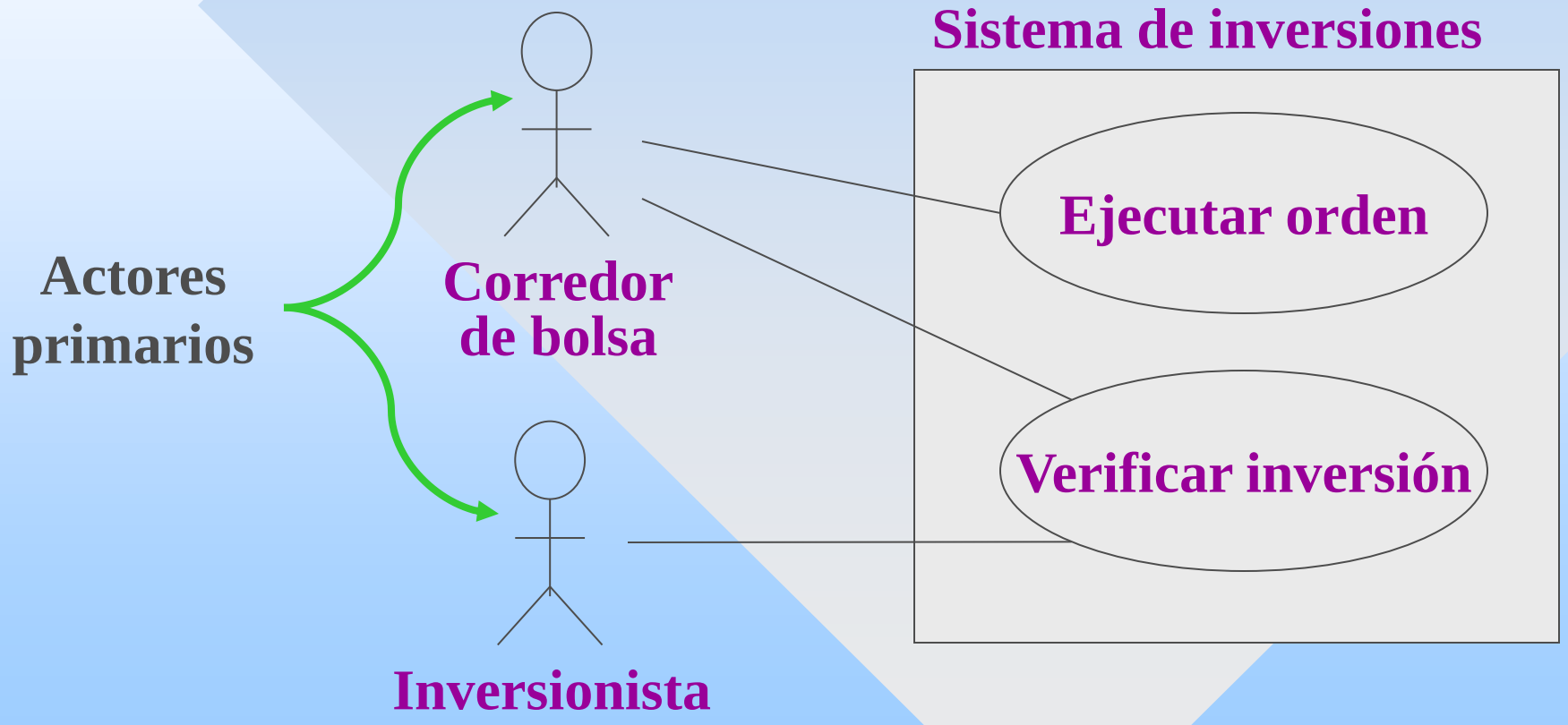


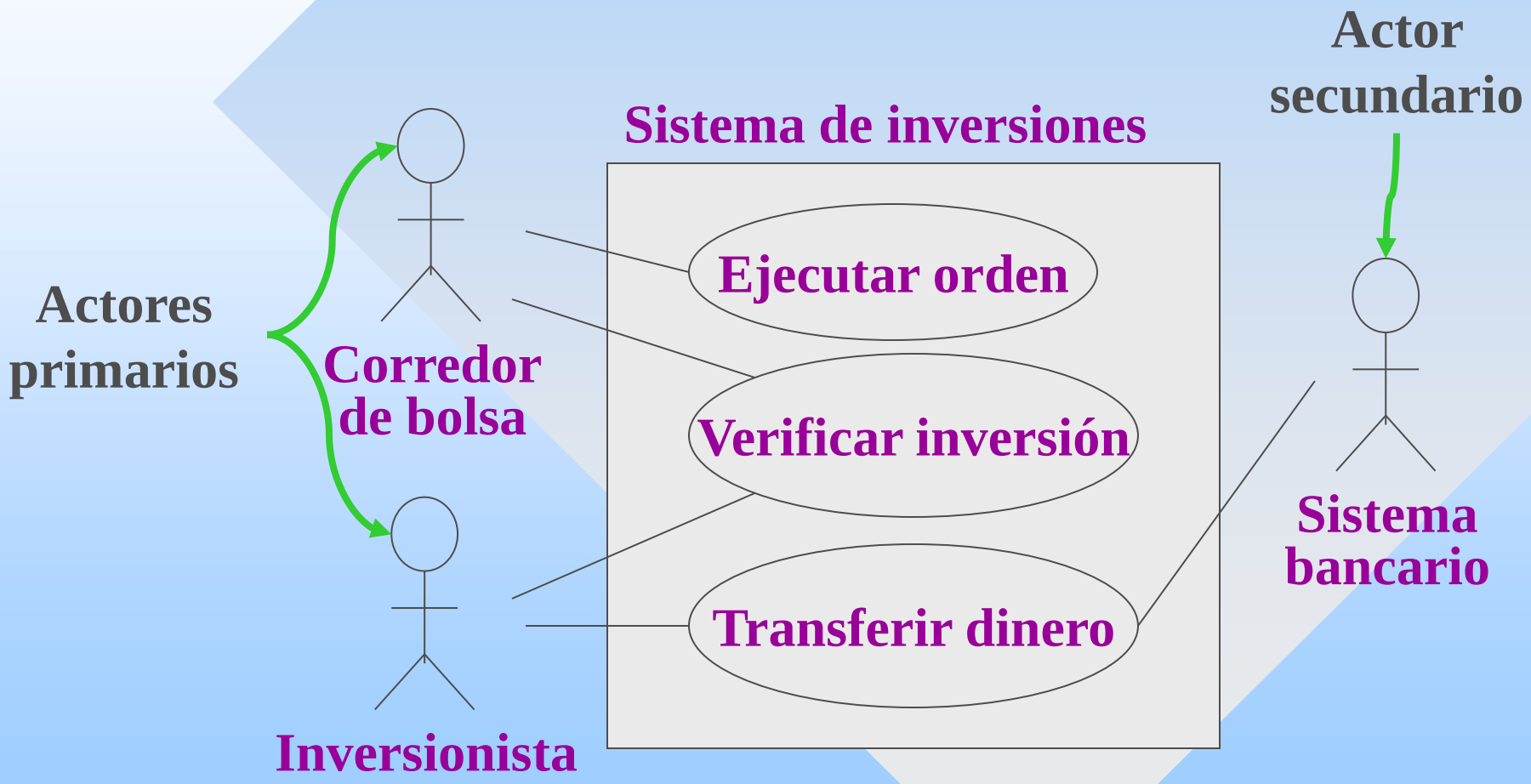
Diagrama básico de un caso de uso



Actores primarios en un diagrama de casos de uso



Actores secundarios en un diagrama de casos de uso

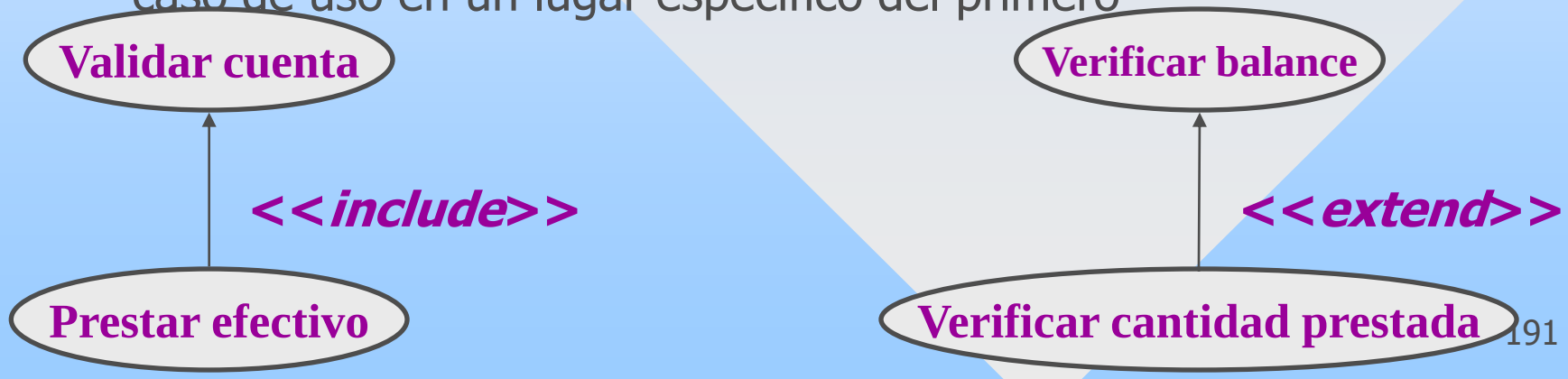


Reglas para crear diagramas de casos de uso

- Los actores primarios se muestran a la izquierda del diagrama y los secundarios a la derecha
- Un caso de uso debe proporcionar un servicio real al usuario
- Escribir los nombres de los casos de uso dentro de la elipse con el fin de ahorrar espacio
- Mantener los diagramas claros y nítidos
- No poner demasiados casos de uso en un diagrama
  - Diagramas sencillos son mas fáciles de entender

Relaciones de generalización en casos de uso

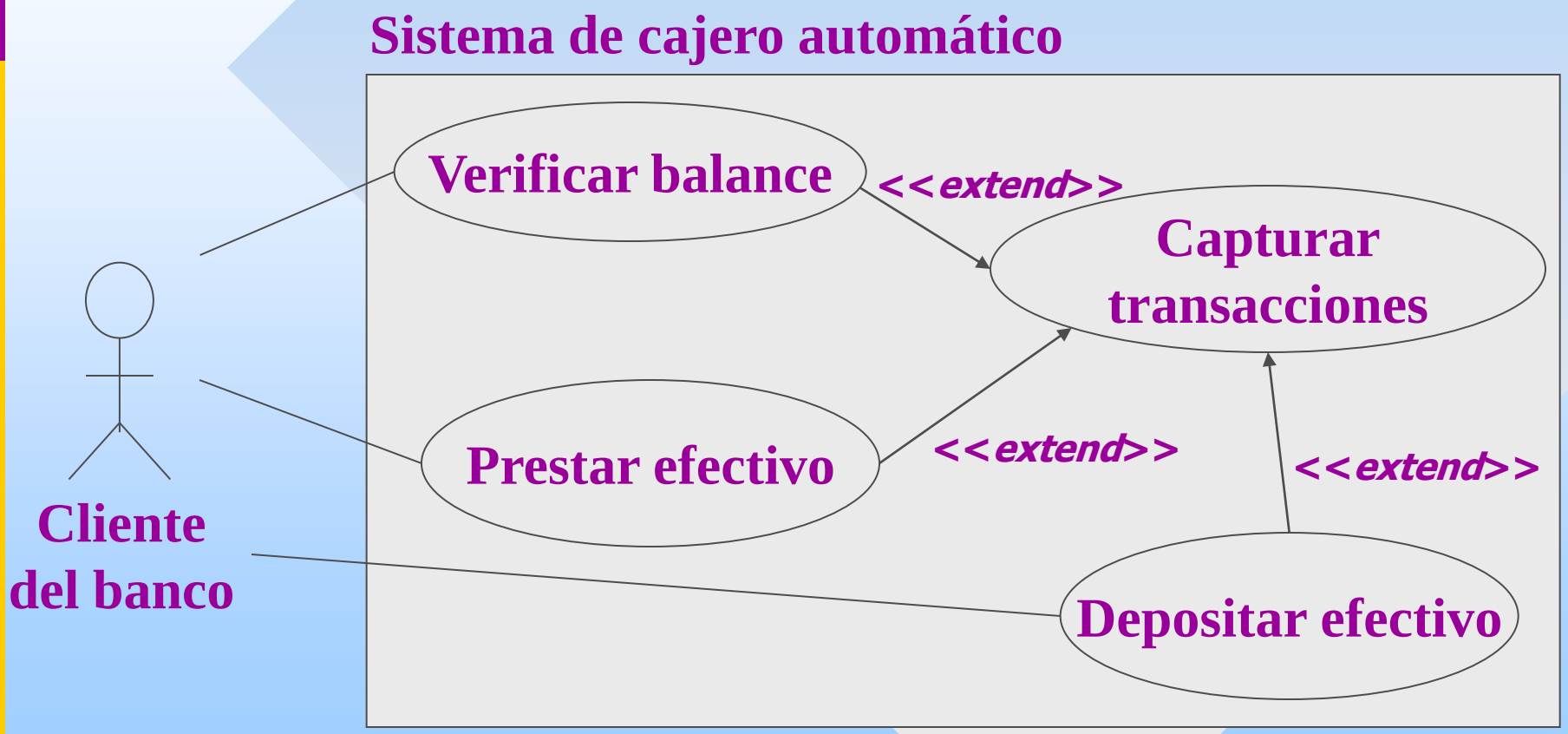
- Estas relaciones están ligadas a los escenarios secundarios
- *Extend*
  - Cuando un caso de uso se extiende a otro al agregar acciones
- *Include*
  - Cuando un caso de uso incorpora el comportamiento de otro caso de uso en un lugar específico del primero



Extensiones de casos de uso (*extend*)

- Las extensiones
  - Permiten al analista cambiar o agregar funcionalidad un caso de uso base, sin alterarlo
    - Un caso de uso es similar a otro, pero este último hace un poco más
  - Se utilizan para modelar partes opcionales de un caso de uso
    - Caminos alternativos complejos se deben modelar en casos de uso diferentes

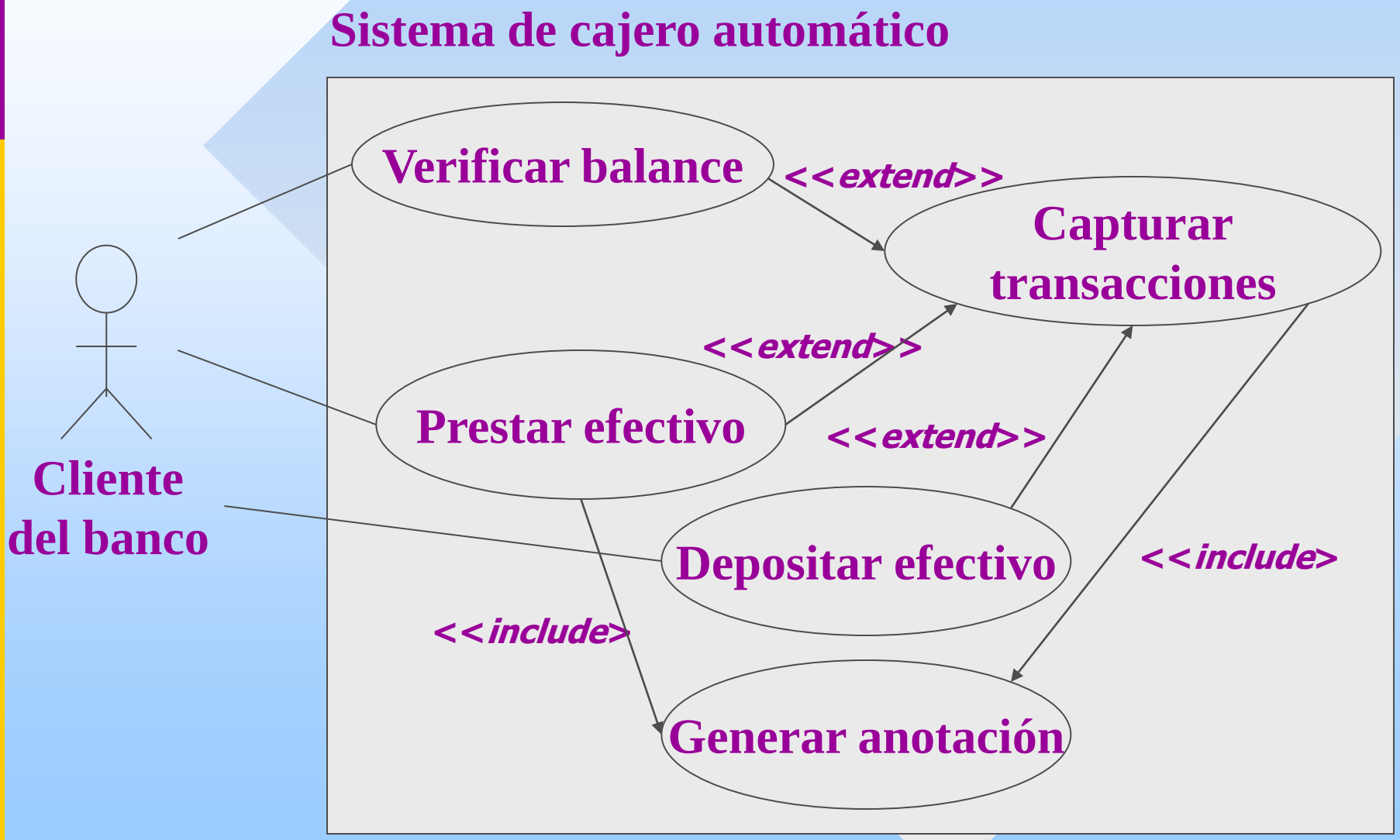
Diagrama de la asociación *extend*



Inclusión de casos de uso (*include*)

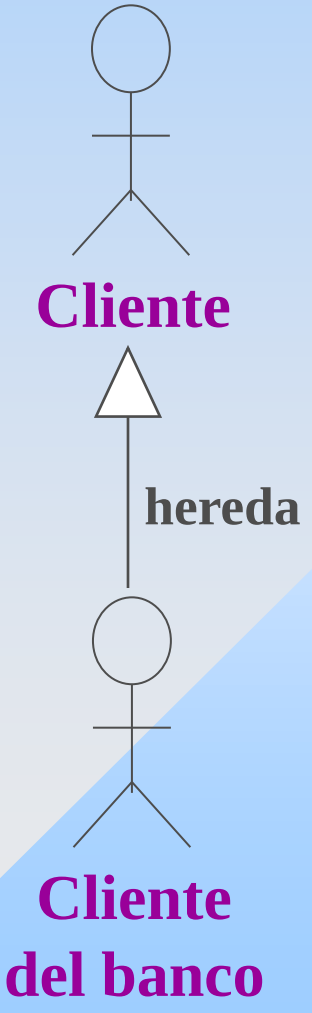
- La inclusión
  - Es diferente que la relación de extensión
  - Es más una subrutina, más que una especialización
  - Aparece cuando un cumulo de acciones es recurrente en muchos casos de usos y se quiere evitar repetir la descripción de este cumulo
    - Cada cumulo de acciones deben manejarse de forma separada en otros casos de uso

Diagrama de la asociación *include*



Herencia en casos de uso y actores

- Existe herencia entre casos de uso
- En un caso de uso la herencia puede se utilizada entre actores
  - Debido a que son considerados como clases en UML
- Herencia entre actores significa que un actor cumple el mismo papel que otro actor y puede jugar papeles adicionales



Propiedades de la herencia entre actores

- Simplifica el diagrama sin perdida de detalle
- Señala la relación jerárquica entre actores
- El actor receptor obtiene las capacidades del actor padre
- Los actores padres no notan la presencia de la herencia

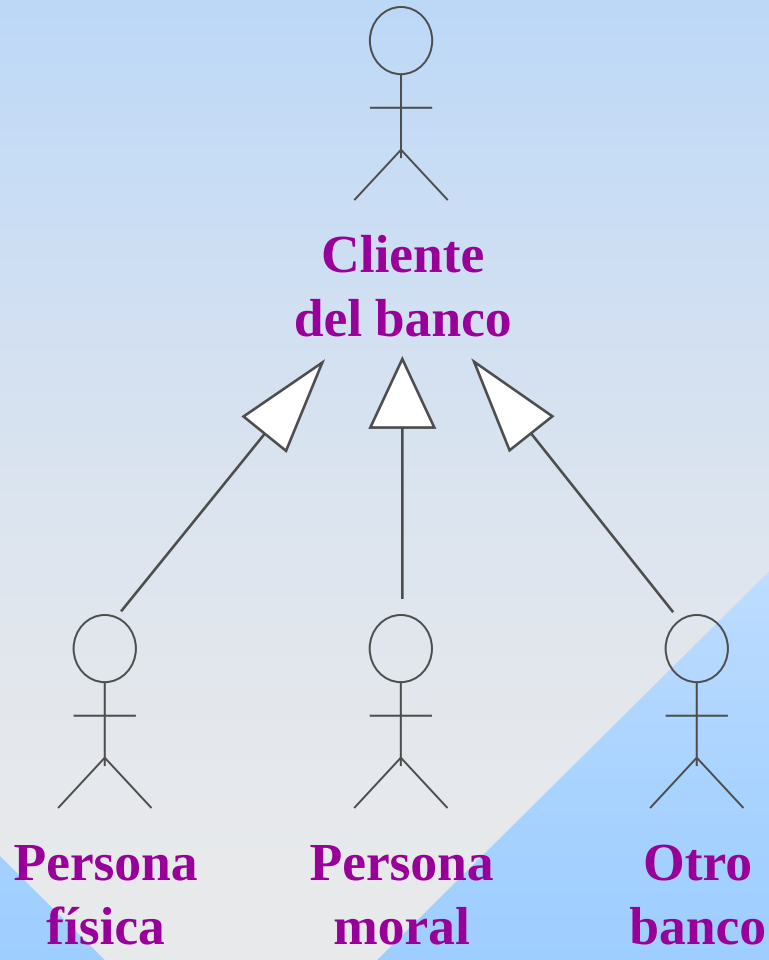
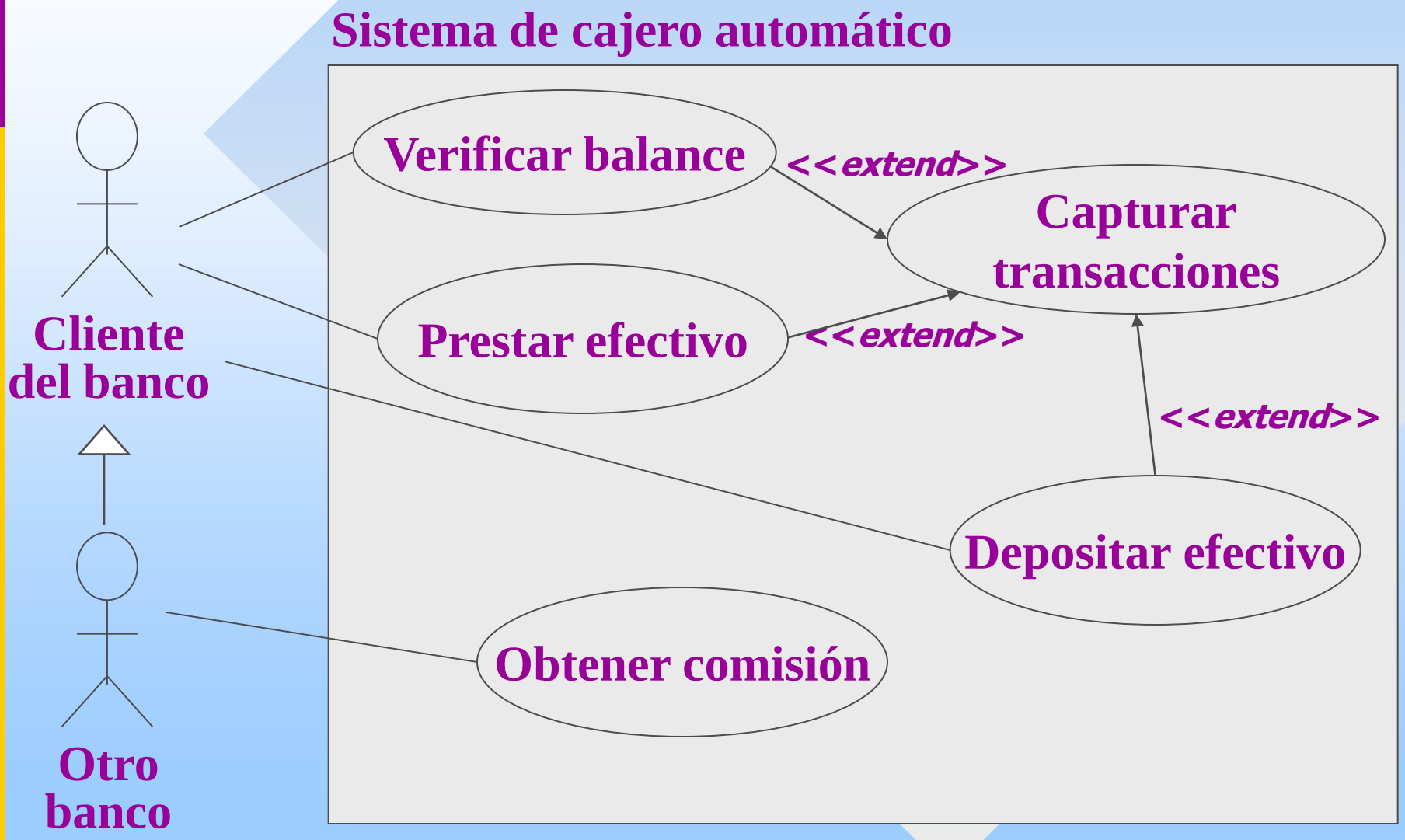
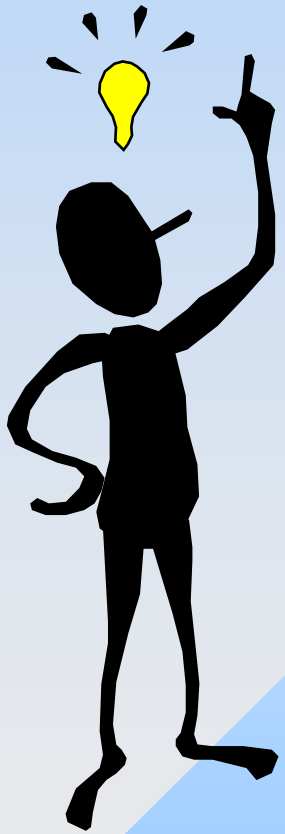


Diagrama de la herencia entre actores



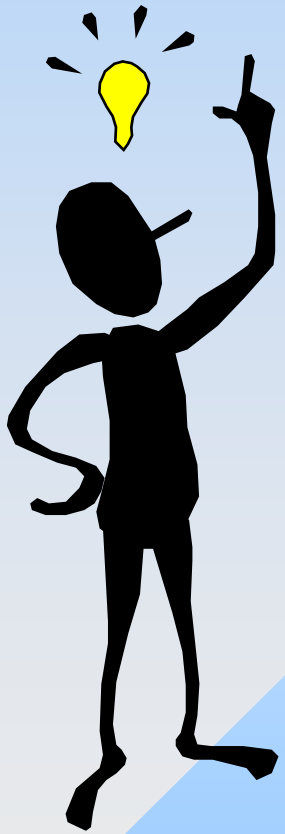
Consejos para el modelado de casos de uso (1)

- Considerar la situación en la que se utiliza el sistema
- Definir las fronteras del sistema
- Identificar todos los actores
- Identificar todas las tareas para las cuales el sistema se utiliza
- Identifique todos los caminos alternativos



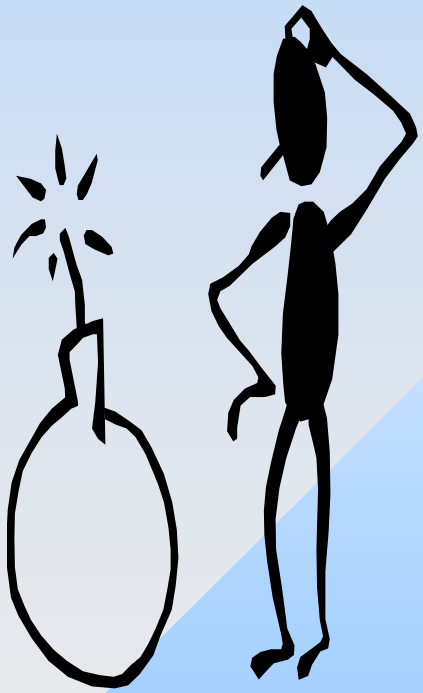
Consejos para el modelado de casos de uso (2)

- Revisar constantemente todos el modelo de caso de uso
- Distinguir los diferentes casos de uso utilizando frecuencias de uso
- Describir únicamente las acciones en las que interviene una interacción entre el actor y el sistema



Ejemplo: el problema

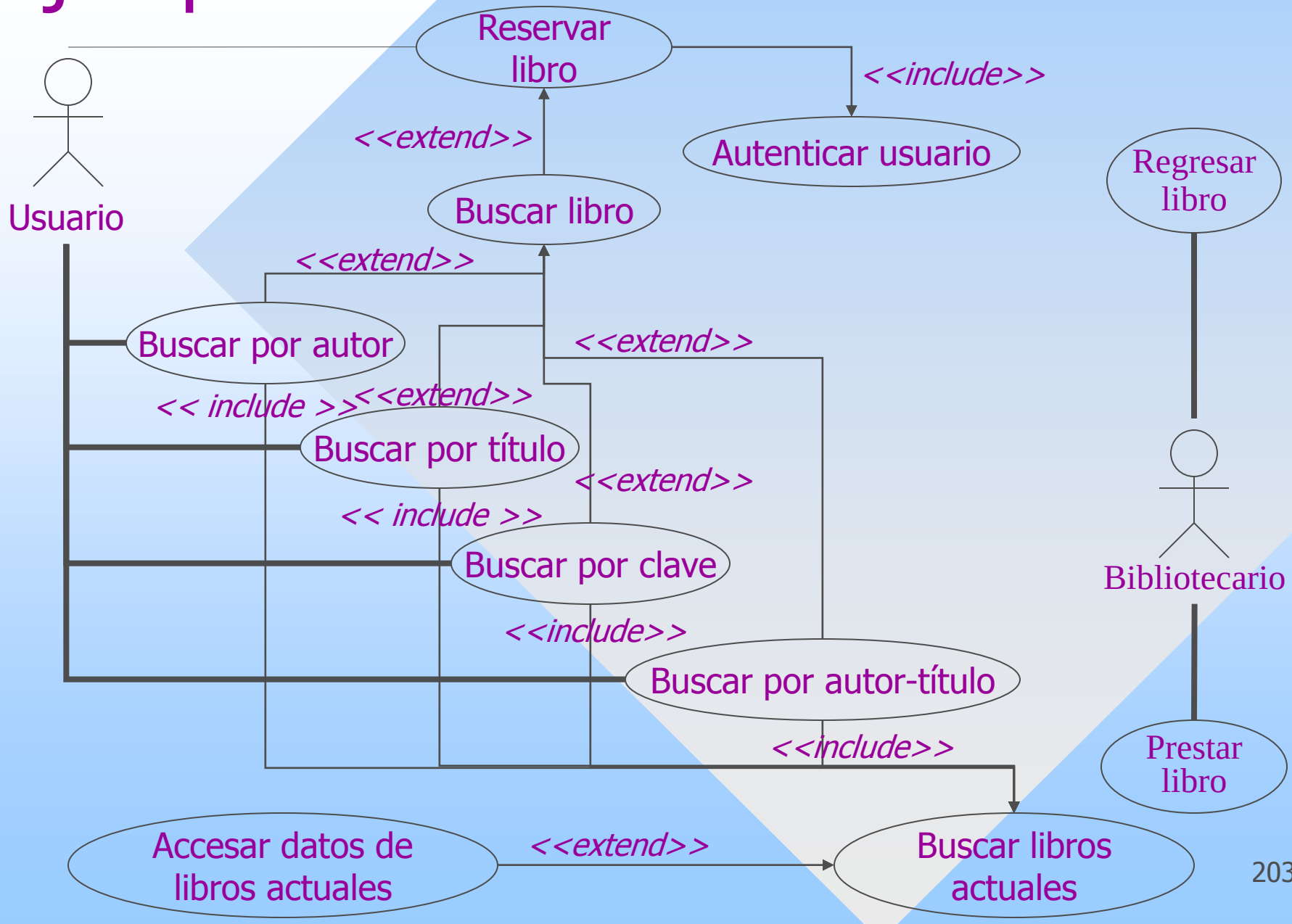
- Definir el modelo de casos de uso para un sistema de biblioteca:
  - Mostrar las relaciones *include* y *extend* entre los casos de uso y los casos de uso abstractos
  - Incluir al menos 3 casos de uso:
    - Caso de uso para la búsqueda de libros
    - Caso de uso para el préstamo de libros
    - Caso de uso para el regreso de libros
  - Definir los caminos normales y alternativos de los casos de uso



Ejemplo: descripción de actores y casos de uso

- Actores
  - Usuario - persona que interactúa directamente con el sistema
  - Bibliotecario - persona que da soporte al sistema y auxilia al usuario en los servicios de préstamo
- Casos de uso
  - Del usuario - buscar por autor, buscar por título, buscar por clave, buscar por autor-título, reservar libro
  - Del bibliotecario - prestar libro
  - Al bibliotecario - regresar libro

Ejemplo: el modelo a documentar



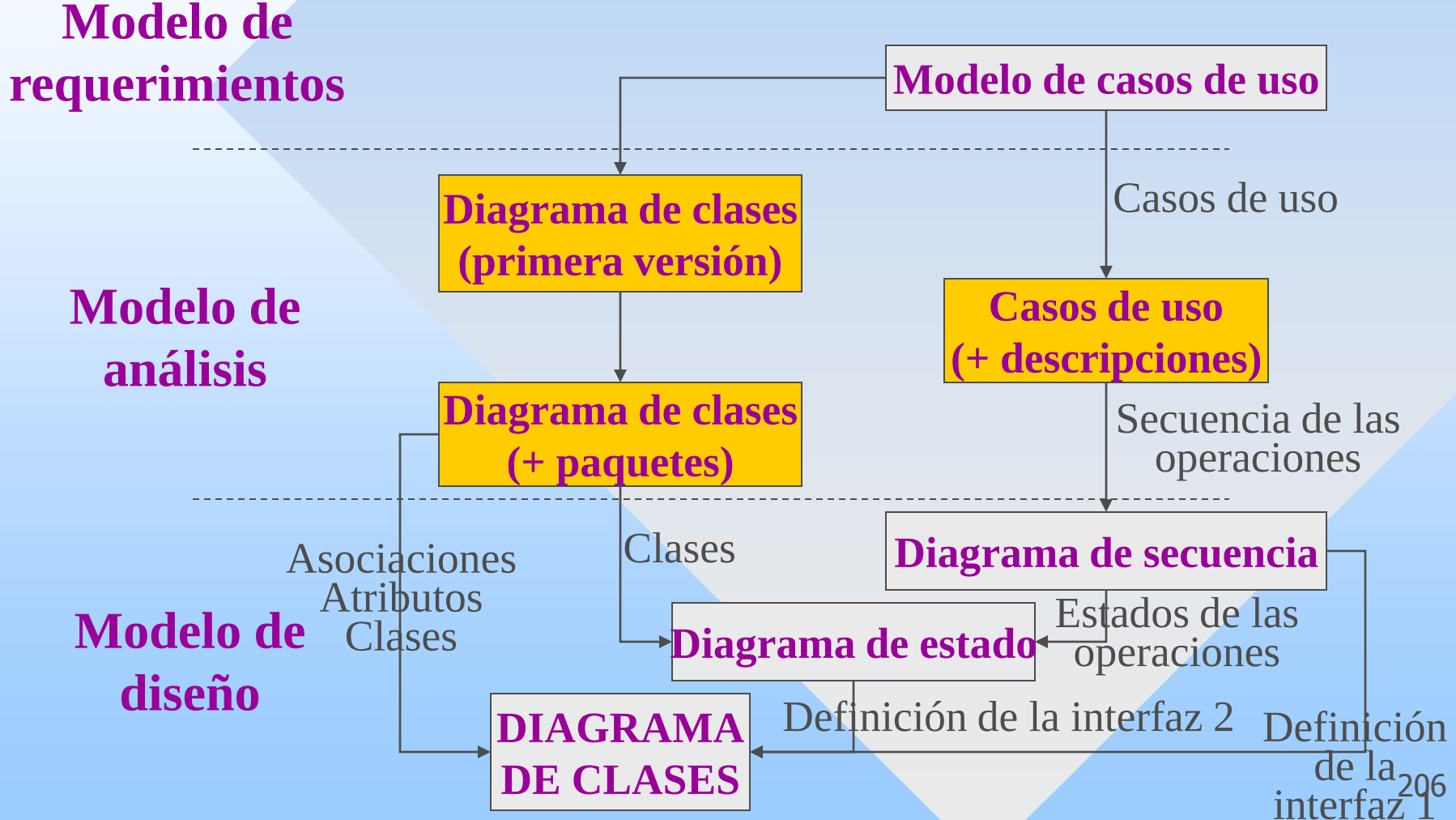
MODELO DE ANÁLISIS (DIAGRAMAS DE CLASE)

Alejandro Domínguez

Panorámica general

- Desarrollar las descripciones de casos de uso
- Describir los principios de los diagramas de clase
- Construcción de diagramas de clase a partir del modelo de casos de uso
- Introducir nuevos tipos de objetos
- Determinar los principios para el diseño por etapas

El modelo de análisis



Producción de un modelo de descripción de casos de uso

INPUTS

- Diagramas de casos de uso

PROCESOS

- Generar descripciones de casos de uso
- Refinar y completar casos de uso y el modelo
- Describir y probar interfaces de usuario
- Describir interfaces del sistema

OUTPUTS

- 1 modelo de casos de uso
- Descripciones del caso de uso
- Descripciones de la interfaz del usuario

Descripciones de casos de uso

- A cada elipse se debe asociar un texto que describa el caso de uso en mas detalle
  - El texto puede ser resúmenes con palabras clave
- Cada elipse puede incluir condicionales, ramificaciones, y ciclos
- *The rational unified process* tiene un formato básico para describir los casos de uso

Formato básico

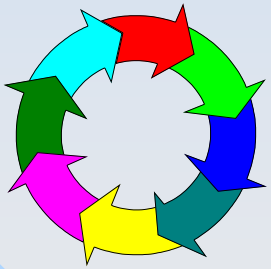
Nombre
Actores
Pre-condiciones
Flujo de eventos
Post-condiciones
Caminos alternativos

Elementos del formato básico de descripción (1)

- Actores
  - Actores relacionados con el caso de uso
- Pre-condiciones
  - Estado del sistema antes que el caso de uso ocurra
- Post-condiciones
  - Estado del sistema después de que el caso de uso ha ocurrido exitosamente

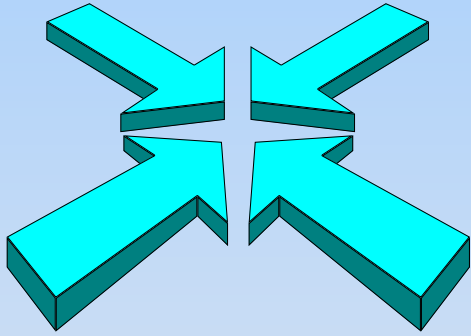
Elementos del formato básico de descripción (2)

- Flujo de eventos
  - Serie de declaraciones que listan los pasos de un caso de uso
  - Son el núcleo principal del caso de uso
  - Asociados a los escenarios primarios
  - Alternativas y repeticiones se muestran utilizando ramificaciones o listandolos en la sección de caminos alternativos

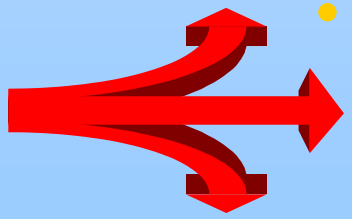


  - Se utilizan las sentencias
    - *Si ... Entonces ...* para indicar alternativas
    - *Repetir ... hasta que ...* cuando se necesita repetir un paso o un conjunto de pasos al menos una vez
    - *Mientras que ...* cuando se necesita repetir un paso o un conjunto de pasos al cero más veces
    - *Para (#inicial) hasta (#final) hacer ...* Cuando se necesita repetir un paso o un conjunto de pasos un número definido de veces

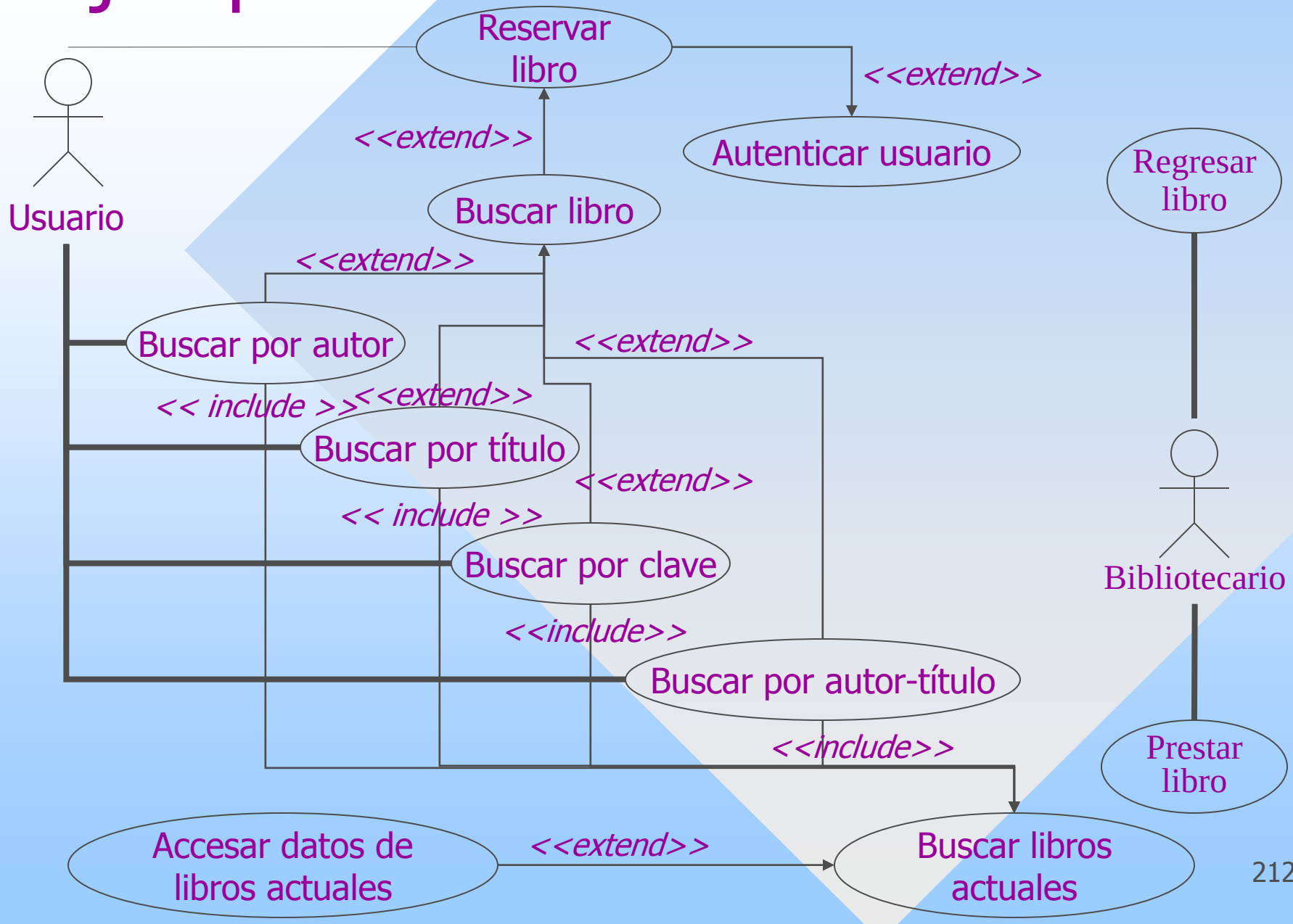
Elementos del formato básico de descripción (3)



- Caminos alternativos
  - Alternativas al camino principal
  - Se utilizan en lugar de una ramificación cuando la alternativa es compleja
  - Apropiados para mostrar cosas que pueden pasar en cualquier momento
    - Algo que puede interrumpir el flujo normal de eventos
  - Excepciones y situaciones de error que pueden ocurrir en el contexto del caso de uso
    - No son errores técnico, sino errores relativos al dominio
      - Pérdida de derechos de acceso, imposibilidad de llenados de formas, etc.



Ejemplo: el modelo a describir



Ejemplo: Descripción de "buscar por título"

Actores: Usuario

Pre-condiciones: El sistema está en espera de datos de búsqueda

Flujo de eventos:

1. El usuario teclea las palabras clave y entonces presiona el botón de búsqueda
2. El sistema despliega los datos del libro que tiene todas las palabras clave que aparecen en el título

   2.1. *do include* "accesar datos de libros actuales"

3. Si el libro existe, entonces *do extend* "buscar libro"

Post-condición: El sistema está en posibilidades de reservar el libro

Ejemplo: Descripción de "reservar libro" (1)

Actores: Usuario

Pre-condiciones: El sistema está en posibilidades de reservar libro

Flujo de eventos:

1. Si el sistema no tiene información para autenticar al usuario, entonces *do extend* "autenticar usuario"
2. Si el usuario conoce datos del libro y desea reservarlo, entonces introduce datos al sistema y presiona el botón reservar, de otra forma el sistema le informa que el libro no está disponible

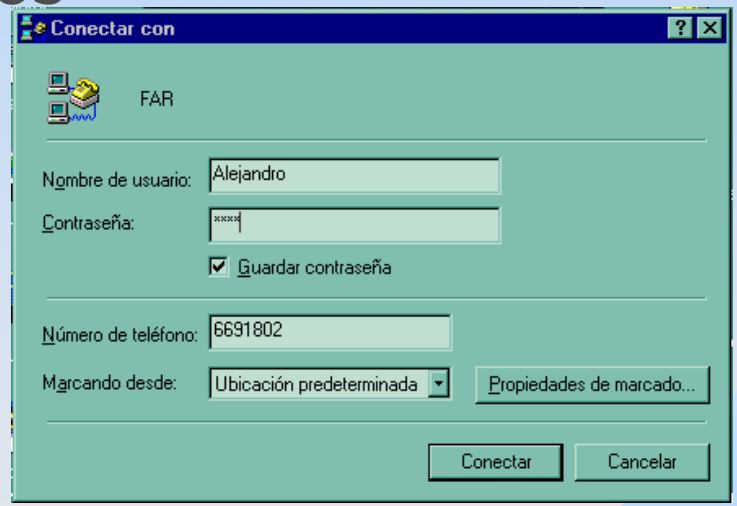
Ejemplo: Descripción de "reservar libro" (2)

4. Si el usuario quiere salir, entonces presiona el botón SALIDA
5. Si el usuario no conoce datos del libro y desea reservarlo, entonces ir casos de uso de búsqueda

Post-condiciones: El sistema hace reservación

Interfaces

- Pueden ser definidas por actores, casos de uso, o ambos
- Dice lo que se espera que la entidad haga
- No es parte de los actores o casos de uso
  - Es una descripción de cómo interactuar con el actor o caso de uso
- Puede existir más de una interfaz para un actor o caso de uso



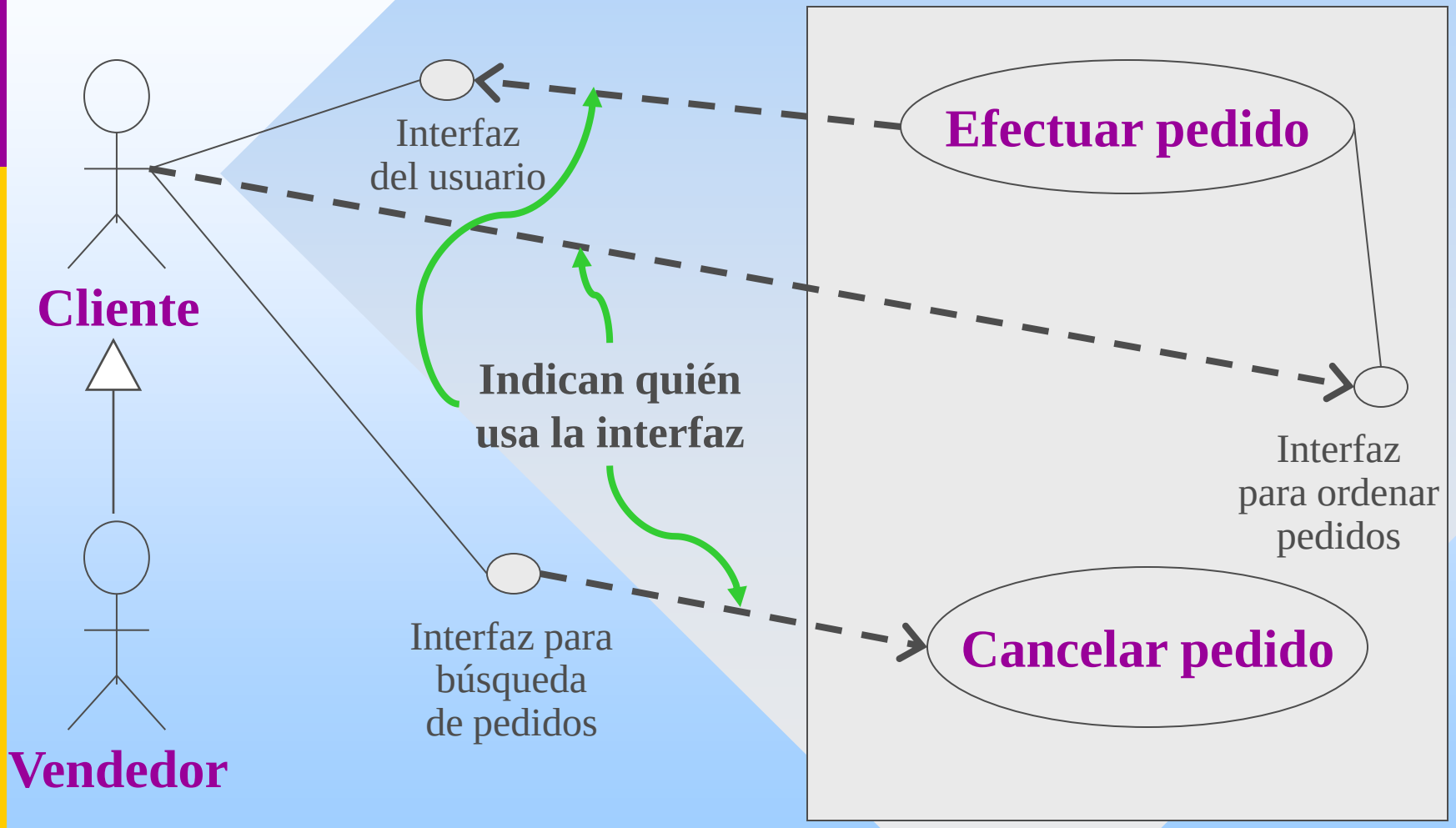
Estructura de las interfaces

- Una interfaz tiene un nombre y un conjunto de firmas de operaciones (*operation signatures*)
  - Una firma de operación señala el tipo de dato que se pasa con operación y el tipo de dato que se pasa cuando la operación se completa
    - La operación señala que se espera que la entidad haga

Ejemplo: Interfaces para el caso de uso "buscar por título"

Buscar por título ( )
Introducir palabras clave (Palabras Clave)
Regresar títulos con palabras clave
Presionar buscar libro( )

Notación para interfaces



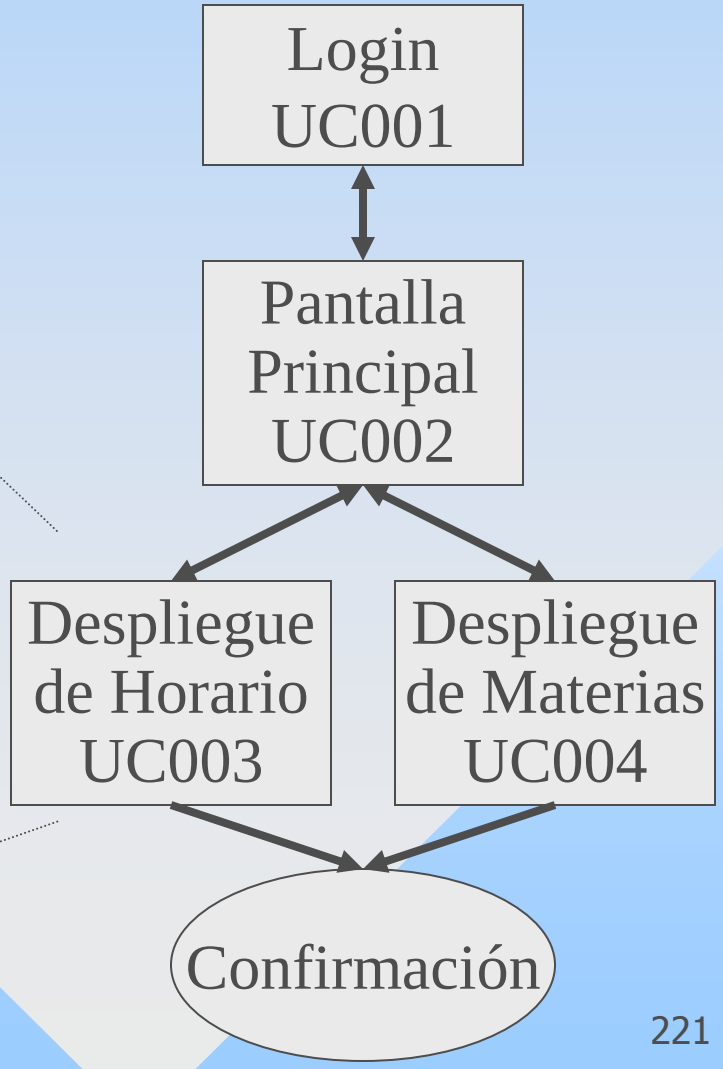
Pantallas para interfaces

- Dada la definición de las interfaces se diseñan las respectivas pantallas
  - Omitir pantallas de “error” y de “información”
  - No invertir demasiado tiempo en la apariencia

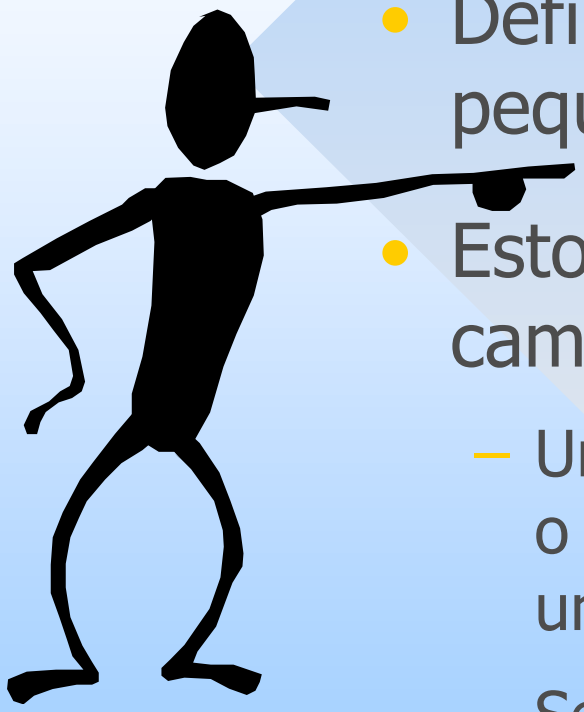
Mapa de navegación de interfaces (1)

- Una vez diseñadas las interfaces se debe hacer un mapa de navegación entre ellas
- El mapa de navegación señala la secuencia de la interacción humano-máquina vía las interfaces de usuario
- Notación
  - Rectángulo = pantalla
    - Señala que otra pantalla será invocada
  - Circulo = pantalla hoja
    - Señala que ninguna otra pantalla será invocada
  - Cada rectángulo y circulo contienen título y el número o nombre del caso de uso
  - Flecha unidireccional = flujo entre interfaces
  - Flecha bidireccional = cancelar y regresar

Mapa de navegación de interfaces (2)



Recomendación para las interfaces



- Definir un número considerable de pequeñas interfaces
- Esto hace el sistema más flexible para cambios futuros debido a que
  - Una interfaz completa se puede agregar o remover fácilmente, más que cambiar una existente
  - Se puede utilizar una misma interfaz con un actor o caso de uso
  - Se pueden reutilizar

Documentación final del sistema por casos de uso (*template*)

1. **Nombre del sistema**
2. **Breve descripción del sistema**
3. **Actores y casos de uso**
4. **Casos de uso**
    - 4.x. **Nombre del Caso de Uso**
        - **Actores**
        - **Pre-condiciones**
        - **Flujo de eventos**
        - **Post-condiciones**
        - **Caminos alternativos**
        - **Interfaces**
5. **Diagrama de casos de uso**
6. **Diagrama de secuencias de interfaces**

El modelo final de casos de uso

FASES DE PRODUCCIÓN

El modelo final de casos de uso (continuación)

FASES DE PRODUCCIÓN

Producción de un diagrama de clases

Propósito: Proporcionar un modelo lógico del software del sistema

INPUTS

- Modelo de casos de uso y descripciones
- Listado de objetos en el dominio del problema

PROCESOS

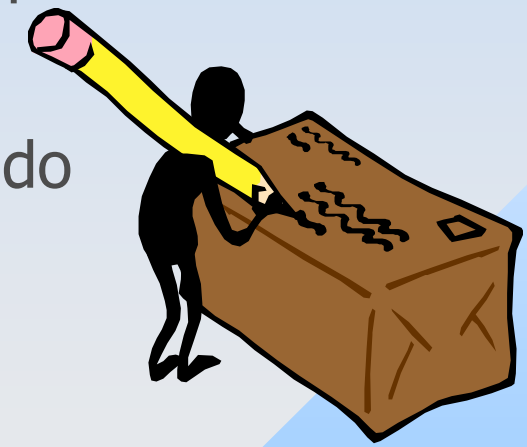
- Bosquejar el diagrama de clases inicial
- Reexaminar el comportamiento de en los casos de uso y los objetos
- Refinar el diagrama de clases
- Efectuar verificaciones de los diagramas
- Revisar diagramas de clase
- Agrupar clases en paquetes

OUTPUTS

- Diagramas de clase incluyendo descripciones de las clases
- Asociaciones entre clases
- El papel que juegan las clases
- Jerarquías de herencia

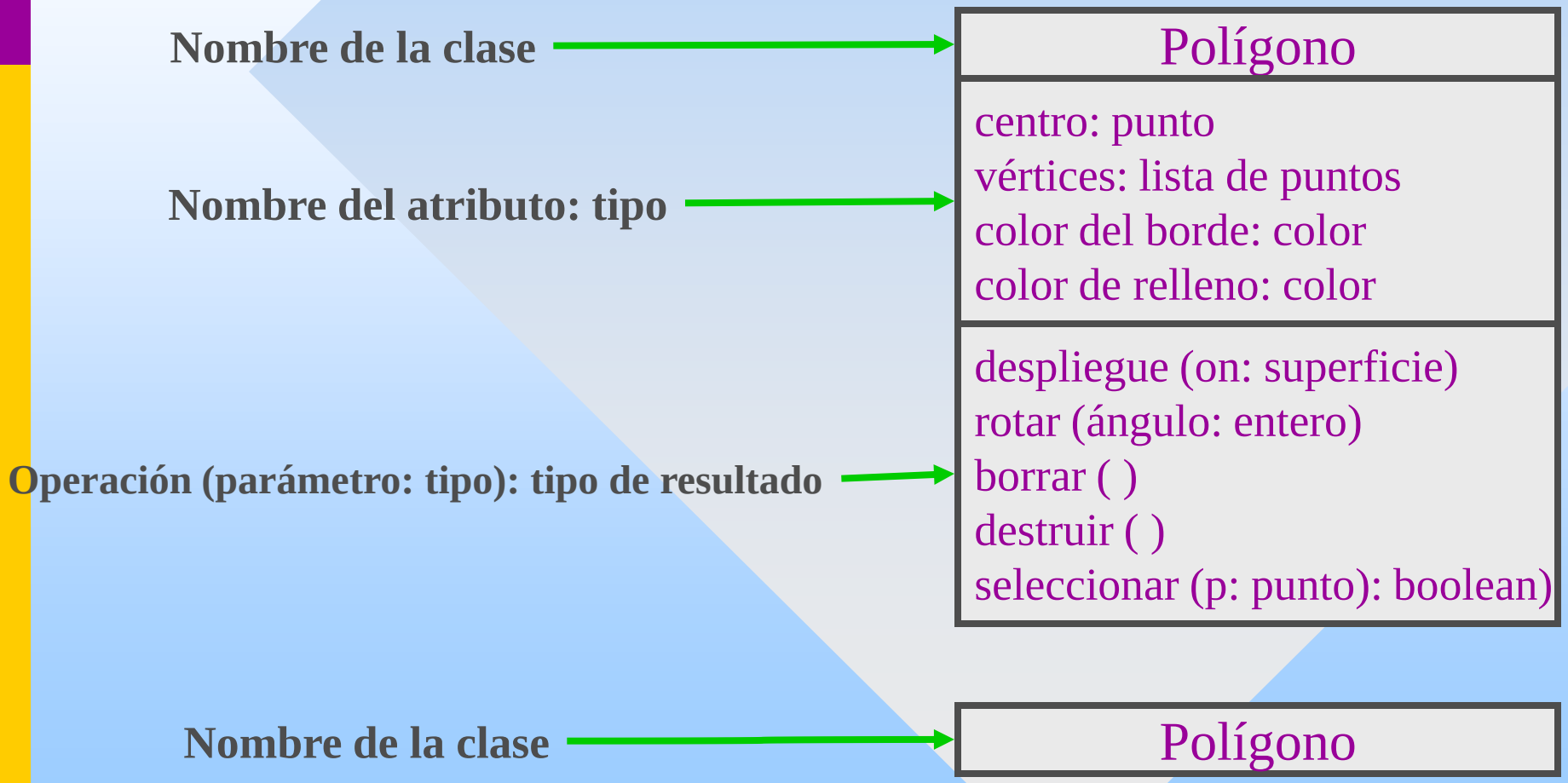
Descripción de objetos atributos y operaciones

- En las descripciones de los casos de uso
  - Los objetos y las clases se determinan subrayando cada sujeto o sustantivo
  - Los atributos se determinan subrayando todos los adjetivos asociados a sus respectivos objetos
  - Las operaciones se determinan subrayando todos los verbos y frases verbales asociados a los respectivos objetos



Clases en UML

Existen dos notaciones posibles:



Objetos en en UML

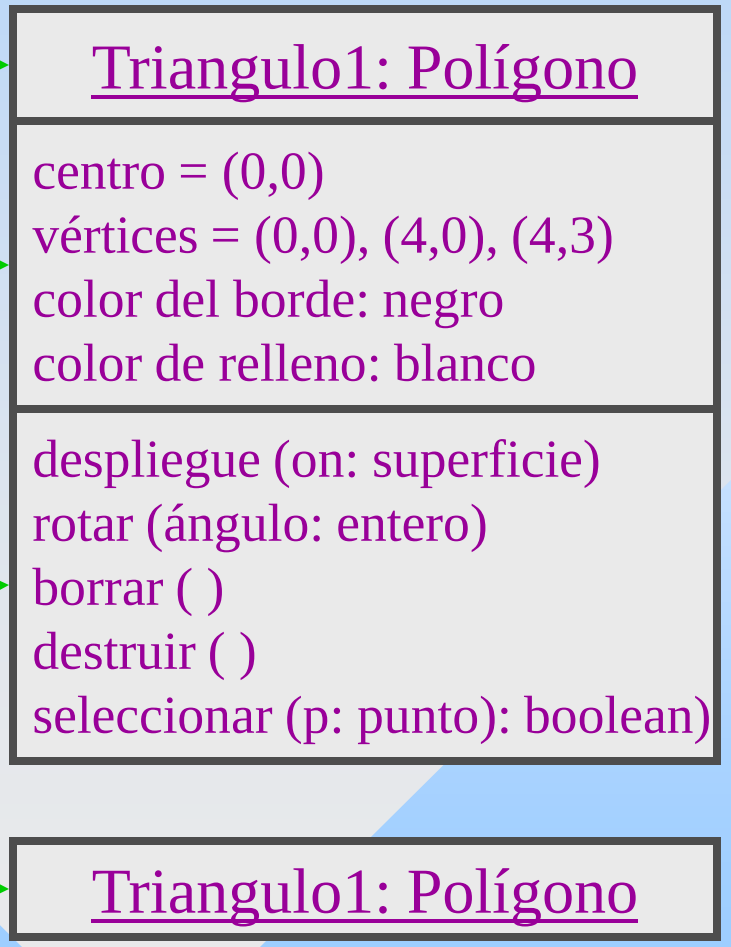
Existen dos notaciones posibles:

Nombre del objeto

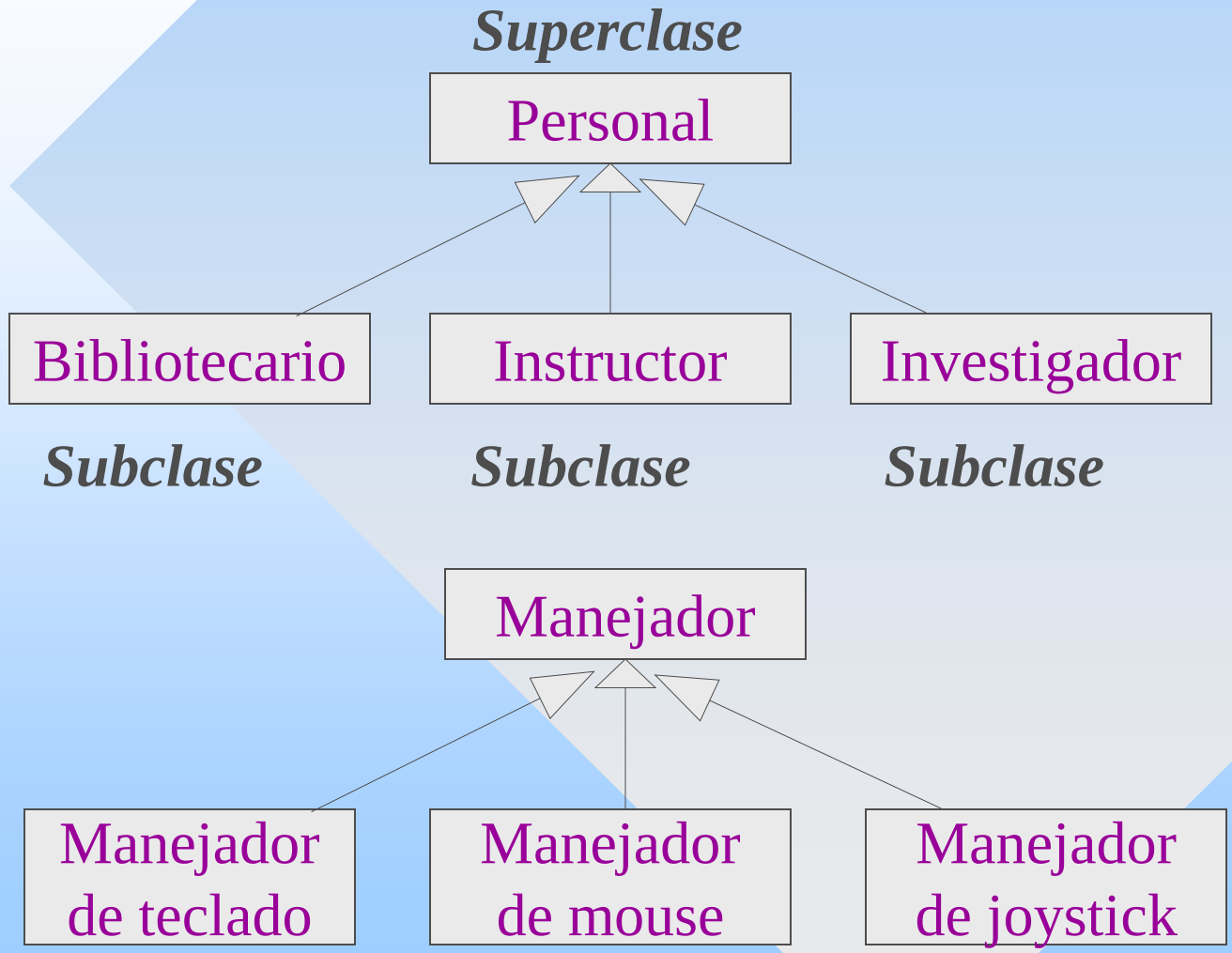
Nombre del atributo: tipo

(mismas operaciones para todas las instancias de una clase)

Nombre del objeto: Nombre de la clase



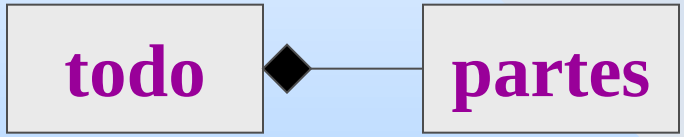
Generalización en UML



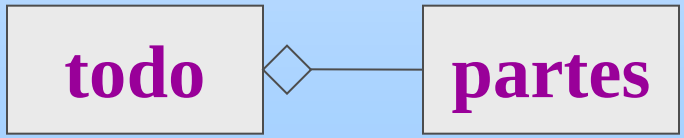
Asociaciones en UML (1)

asociación

navegación (permite comunicación por medio del envío de mensajes)



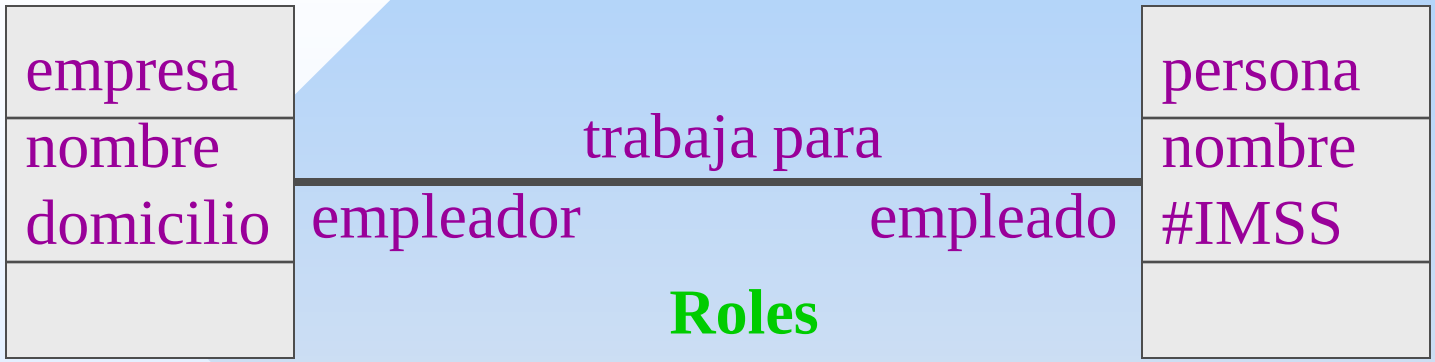
composición (todo "contiene" partes)
"ventana contiene menú, botón, ..."
(el todo posee a sus partes)
(las partes viven en el todo)



agregación (todo "tiene" partes)
"la universidad tiene facultades"
(las partes componen el todo)

Asociaciones en UML (2)

Asociaciones (3)



Muchos → ♦

Exactamente uno → 1

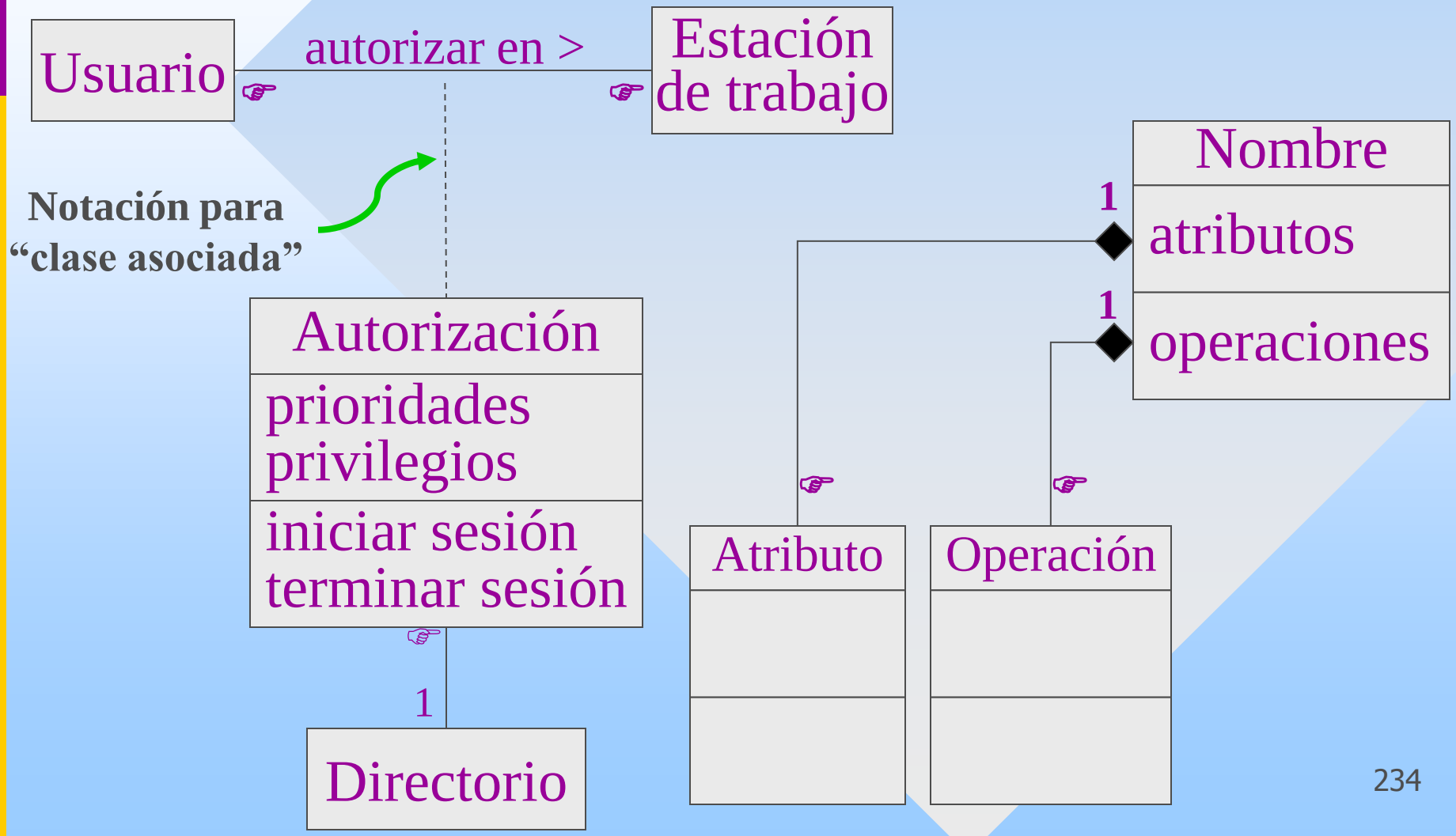
Cero o más → 0..♦

Uno o más → 1..♦

Cero o uno → 0..1

Rango especificado → 2..4, 7

Ejemplos de asociaciones en UML

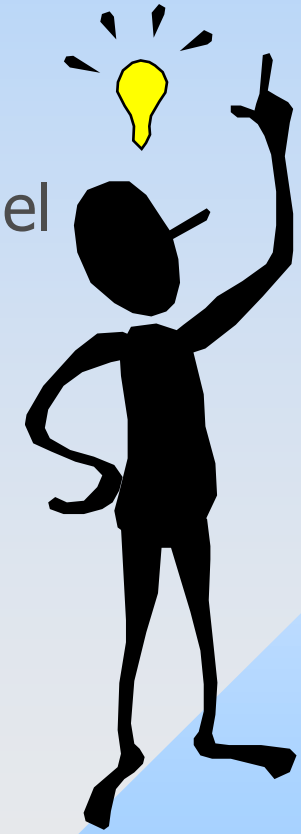


Estereotipos

- Se utilizan para especificar la semántica de elementos ya definidos en UML
- Son elementos generalizables (se pueden especializar o generalizar)
- Se pueden especificar en una clase delante del nombre de la clase
- Indican que tipo de clase es y se indica como << >>
- Se sitúan encima o delante del nombre del elemento

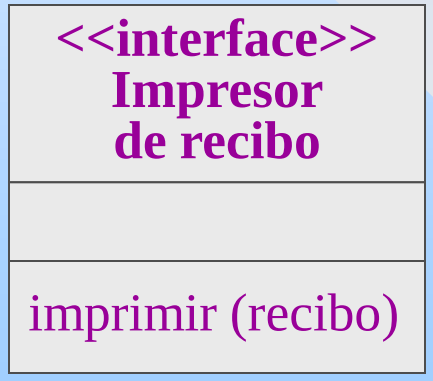
3 estereotipos importantes

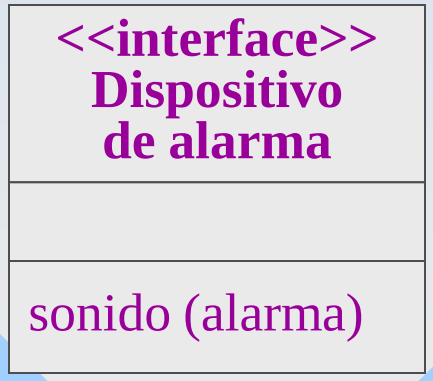
- Clases interfaz
  - Para todo lo relacionado las interfaces del sistema
- Clases entidad
  - Para información persistente y el comportamiento acoplado a ella
- Clases control
  - Para funcionalidad no necesariamente atada a otras clases



Estereotipos: Clases interfaz

- Contienen la funcionalidad directamente dependiente del entorno del sistema
- Se centran en la interacción entre los actores y los casos de uso



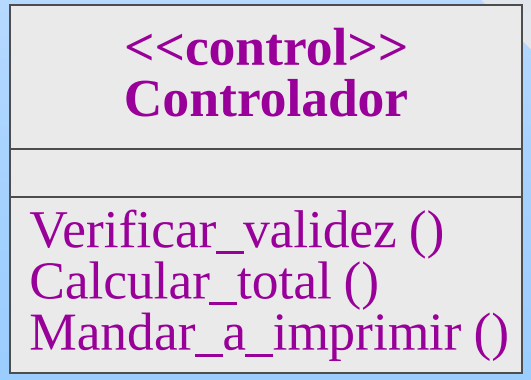


Estereotipos: Clases entidad y sus atributos

- Almacenan información que persiste después de completar un caso de uso
- Definen el comportamiento para manipular esta información
- Las operaciones se caracterizan por:
  - almacenar y obtener información
  - crear y remover objetos
  - el comportamiento cambiante si el objeto entidad cambia

Estereotipos: Clases control

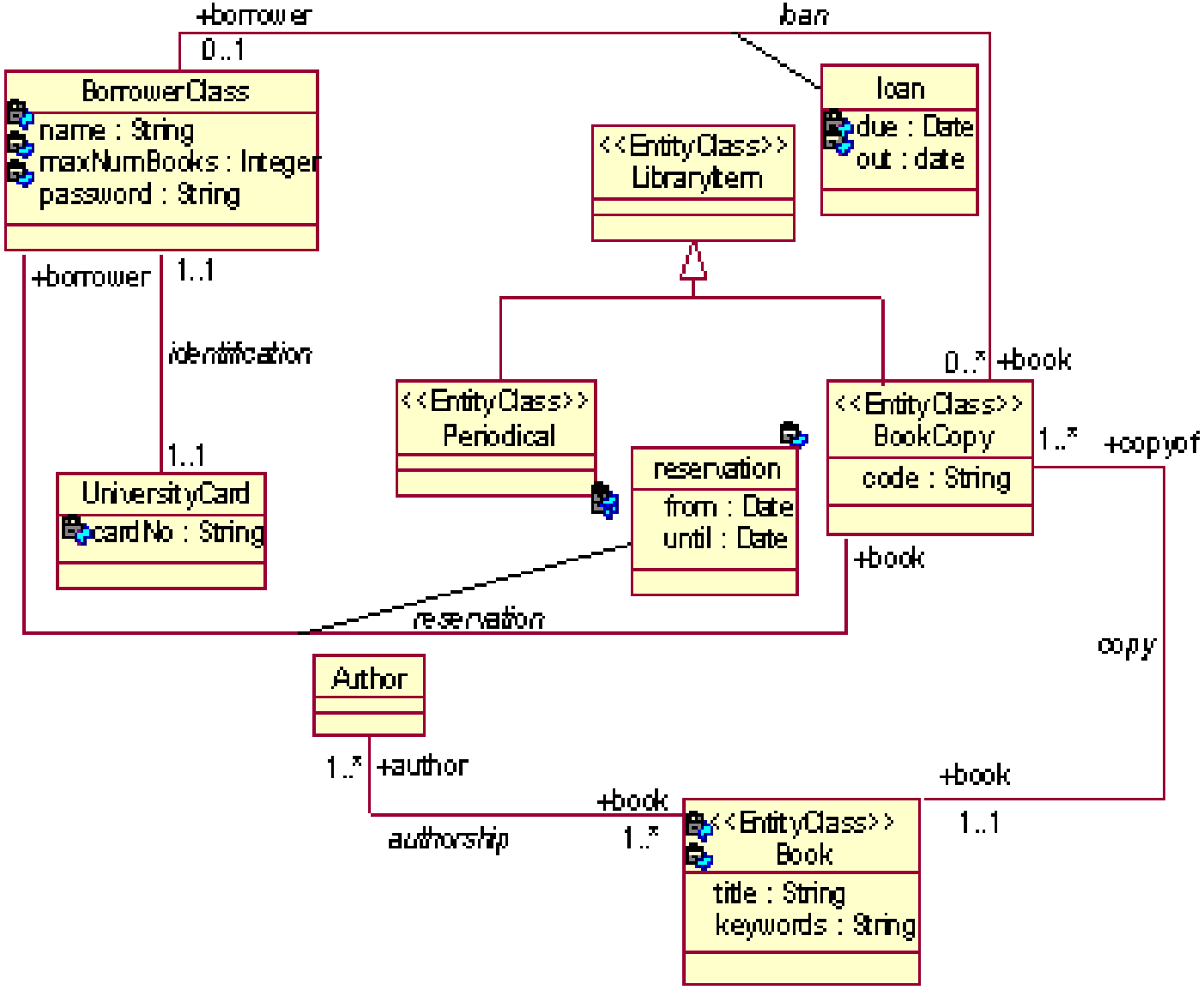
- Son necesarias para:
  - Definir el comportamiento no natural en las clases interface y entidad
  - Fungir como "goma de pegar" entre clases en casos de uso
  - Definir comportamientos típicos de control
  - Mejorar el mantenimiento del sistema



Explotación de los casos de uso para dibujar diagramas de clase

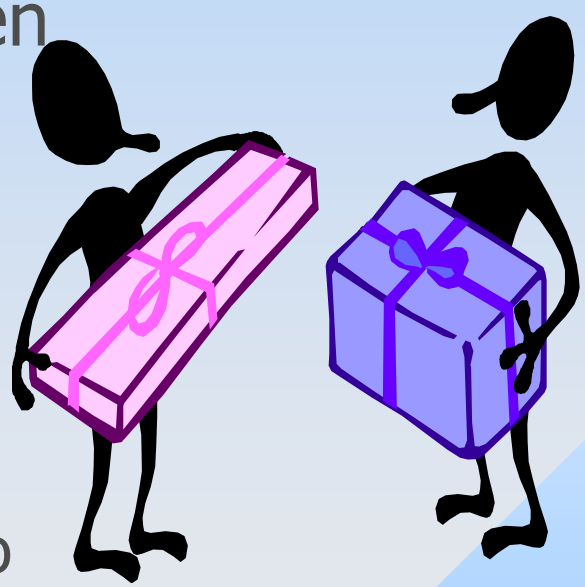
- Emplear ***clases***, ***casos de uso***, ***descripciones de casos de uso***, ***listas de objetos***, ***clases***, ***atributos*** y ***operaciones*** para
  - definir el primer bosquejo de las ***clases*** y los ***atributos***
  - definir ***asociaciones*** entre clases
  - describir ***roles*** y ***responsabilidades*** de cada clase
  - distribuir el ***comportamiento*** especificado en los ***casos de uso*** para cada clase
  - asegurar que existe una ***clase*** para ***cada comportamiento***
  - revisar y refinar el diagrama
  - producir una vista del caso de uso, i.e., un diagrama de clases para cada caso de uso
  - producir un único diagrama de clases para todos los casos de uso

Library System: Analysis Class Diagram



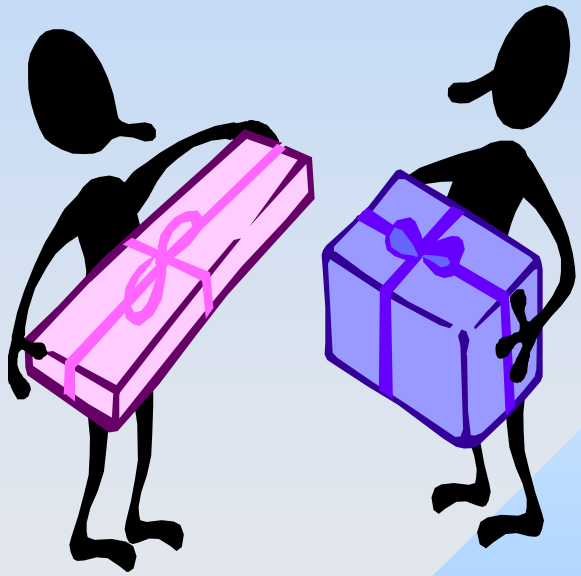
Paquetes (1)

- Son consecuencia del gran número de clases existentes en un sistema
- Son necesarios para:
  - Proporcionar funcionalidad opcional
  - Minimizar los efectos del cambio
- Deben poseer
  - Acoplamiento interno fuerte
  - Acoplamiento externo débil

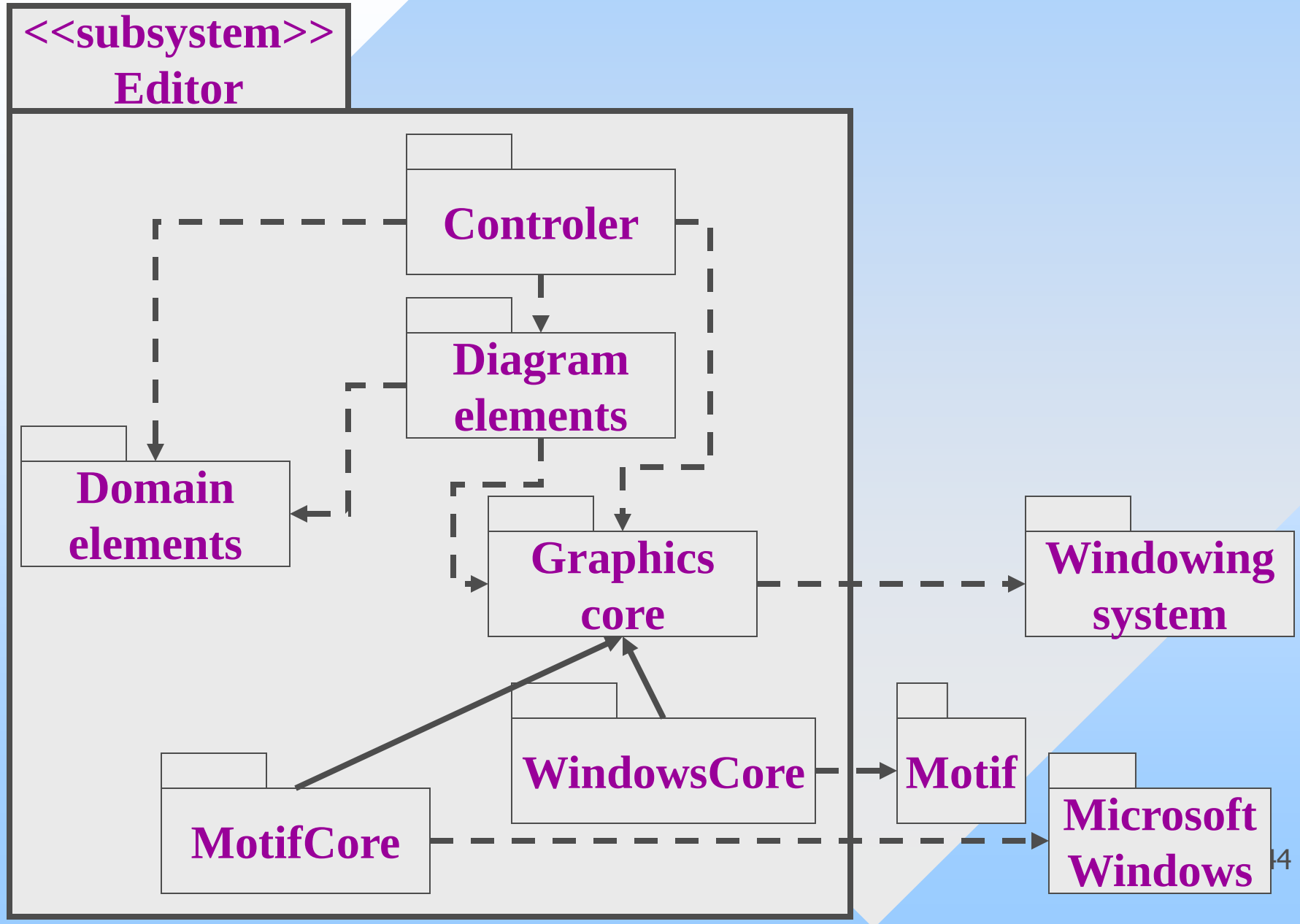


Paquetes (2)

- Pueden
  - Contener paquetes anidados con "paquetes de servicio" como partes atómicas
  - Tener clases individuales fuera de ellos
  - Ser el resultado de problemas organizacionales o administrativos



Ejemplo de paquetes



Modelo de análisis

ETAPAS DE PRODUCCIÓN

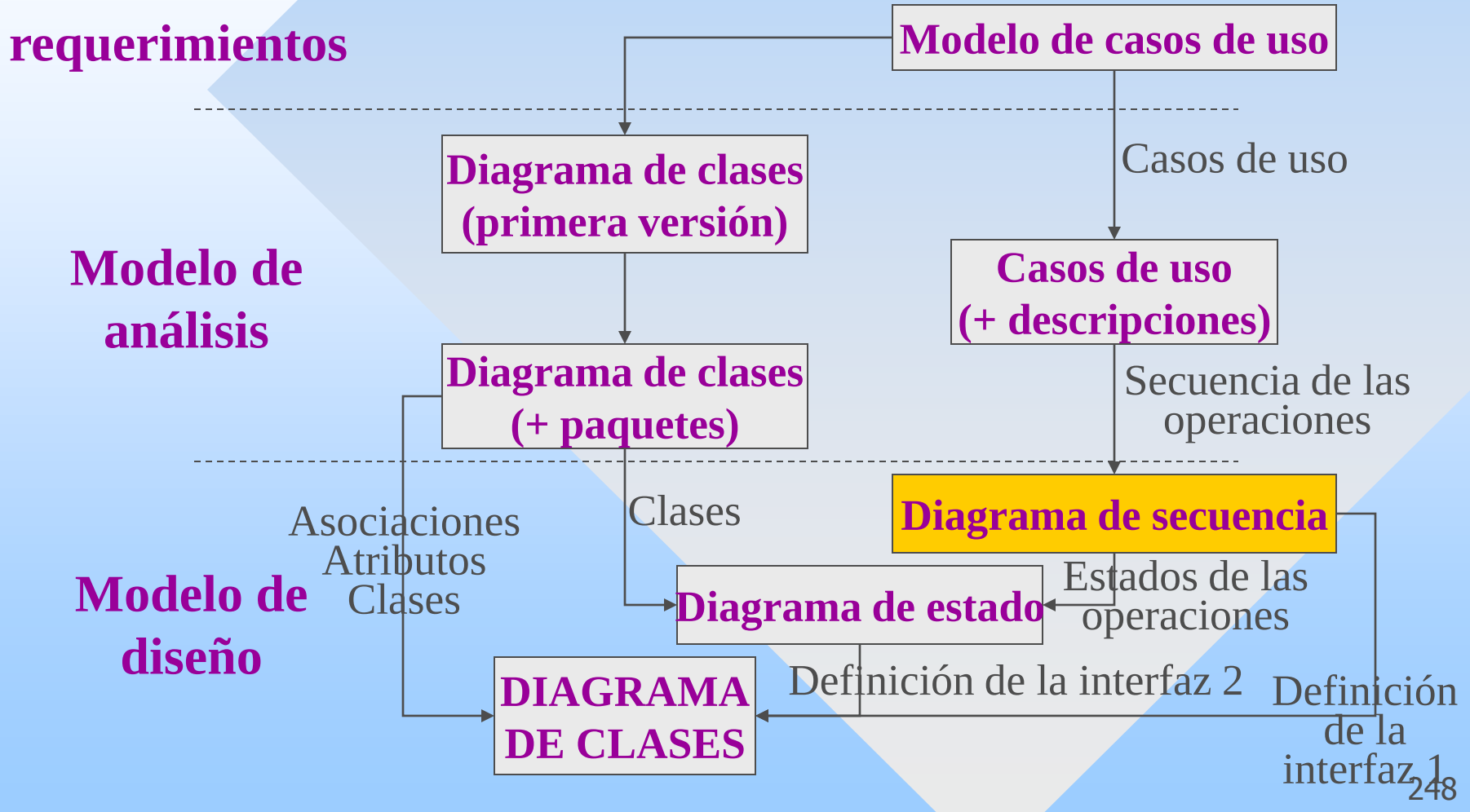
EL MODELO DE DISEÑO (DIAGRAMAS DE SECUENCIA)

Alejandro Domínguez

Panorámica general

- Definir la relación entre el análisis y el diseño
- Definir las fases de diseño
- Señalar el impacto que tiene el ambiente de implementación
- Definir los diagramas de secuencia
- Crear clases interfaz

El modelo de diseño I



Contenido del modelo de diseño

INPUTS

- Especificación de requerimientos relacionados al ambiente de implementación
- Modelo de análisis y diagramas de clase
- Descripciones de los casos de uso

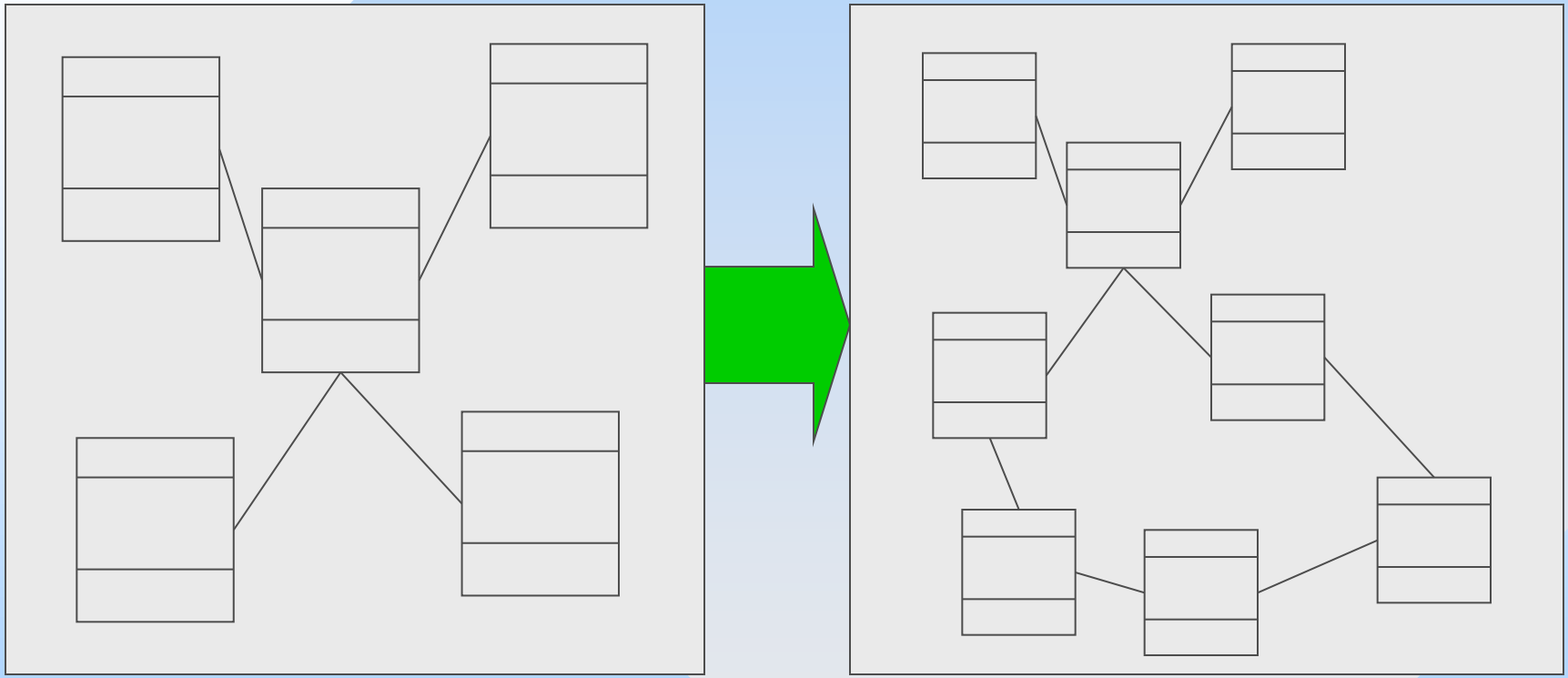
PROCESOS

- Identificar el ambiente de implementación
- Modelar el diseño inicial del diagrama de clases
- Diseñar el control de flujo
- Definir la clase interfaz
- Modelar diagramas de estados
- Finalizar el diseño del diagrama de clases

OUTPUTS

- Diagramas de secuencia (un diagrama por caso de uso)
- Diagramas de estado (un diagrama por clase)
- Modelo de diseño completo (diagramas de clase)

Diagramas de clase con UML en análisis y diseño



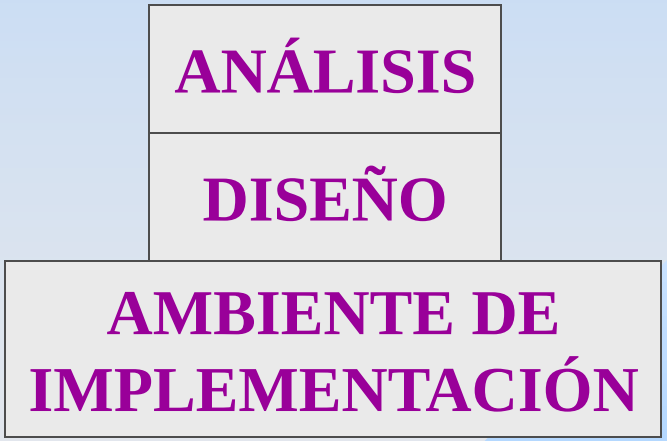
DIAGRAMAS DE CLASE EN EL ANÁLISIS
Modelo conceptual del sistema

DIAGRAMAS DE CLASE EN EL DISEÑO
Construcción del sistema

MISMA NOTACIÓN, DIFERENTE PERSPECTIVA

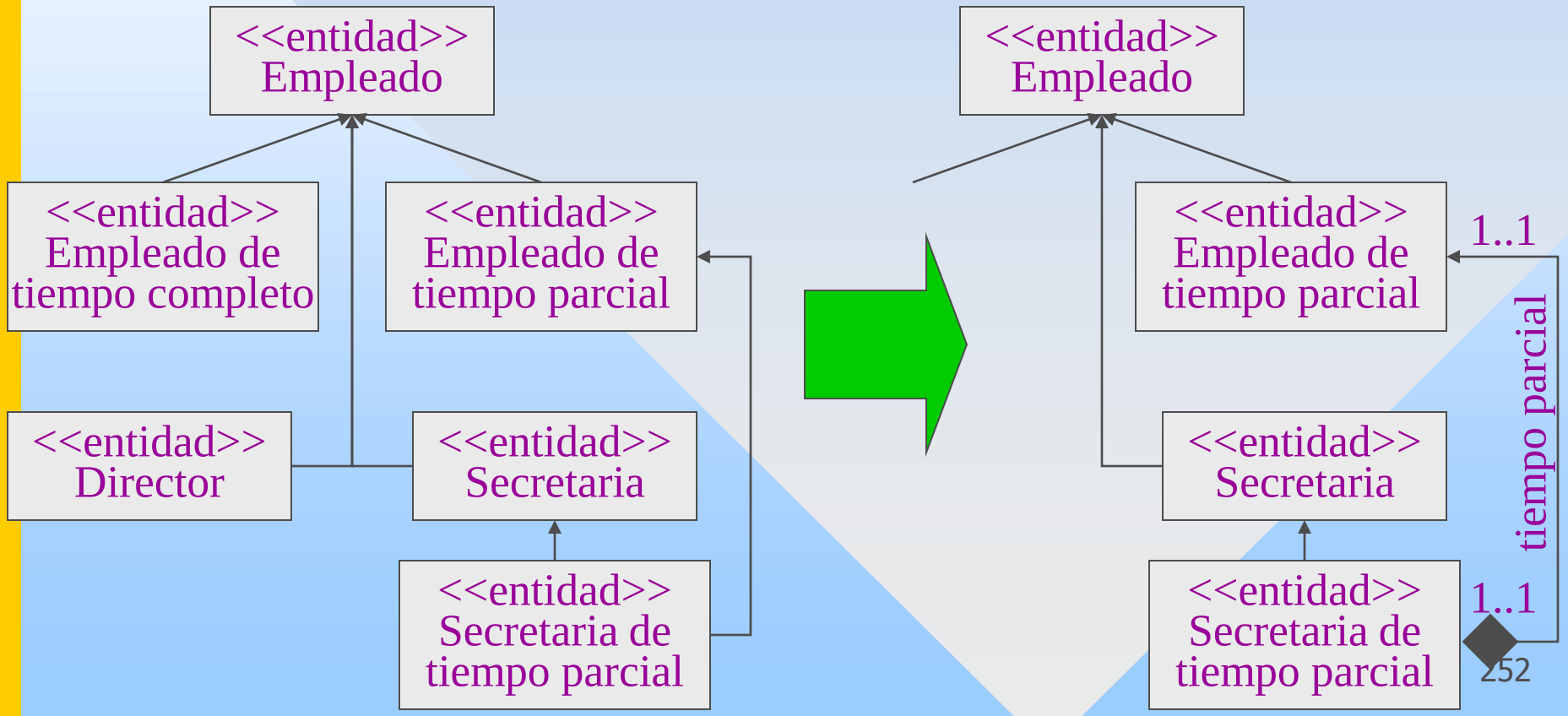
El ambiente de implementación

- Factores que intervienen
  - Sistema operativo
  - Lenguaje de programación
  - Sistema de administración de la interfaz de usuario
  - Administrados de bases de datos
  - Bibliotecas reutilizables
  - Procesos de desarrollo

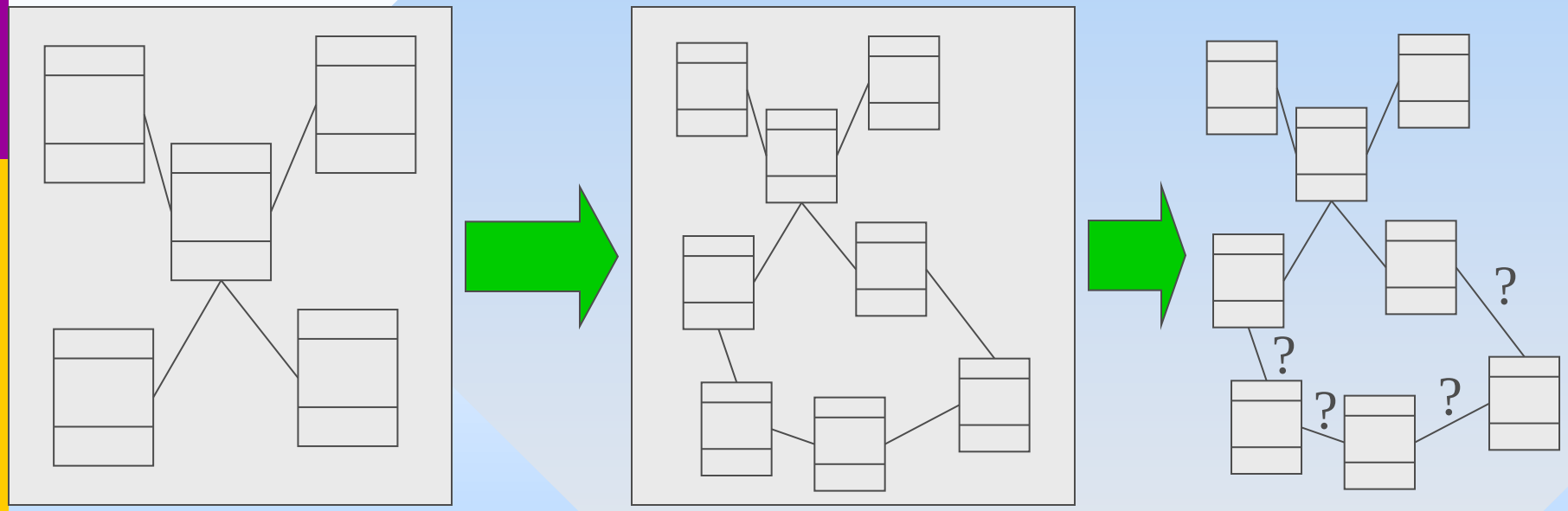


Ejemplo: Cambios debido al lenguaje de implementación

- Utilización de un lenguaje que no soporta herencia múltiple (e.g. Java)



Etapas iniciales del diseño



Traducción del modelo de análisis al modelo de diseño

Diseño del flujo de control

Adaptación al medio ambiente y revisar

Diagramas de secuencia

- De todas la técnicas de modelado de UML, esta es probablemente la más compleja
- Son útiles para decidir y modelar "cómo" el sistema llevará a cabo lo "que" se describió en el modelo de casos de uso
- Ayudan a asignar responsabilidades a las clases y objetos
- Ayudan para que se pueda comprender en un programa OO el flujo de control (secuencia) general de los objetos

Receta para la creación de un diagrama de secuencia (1)

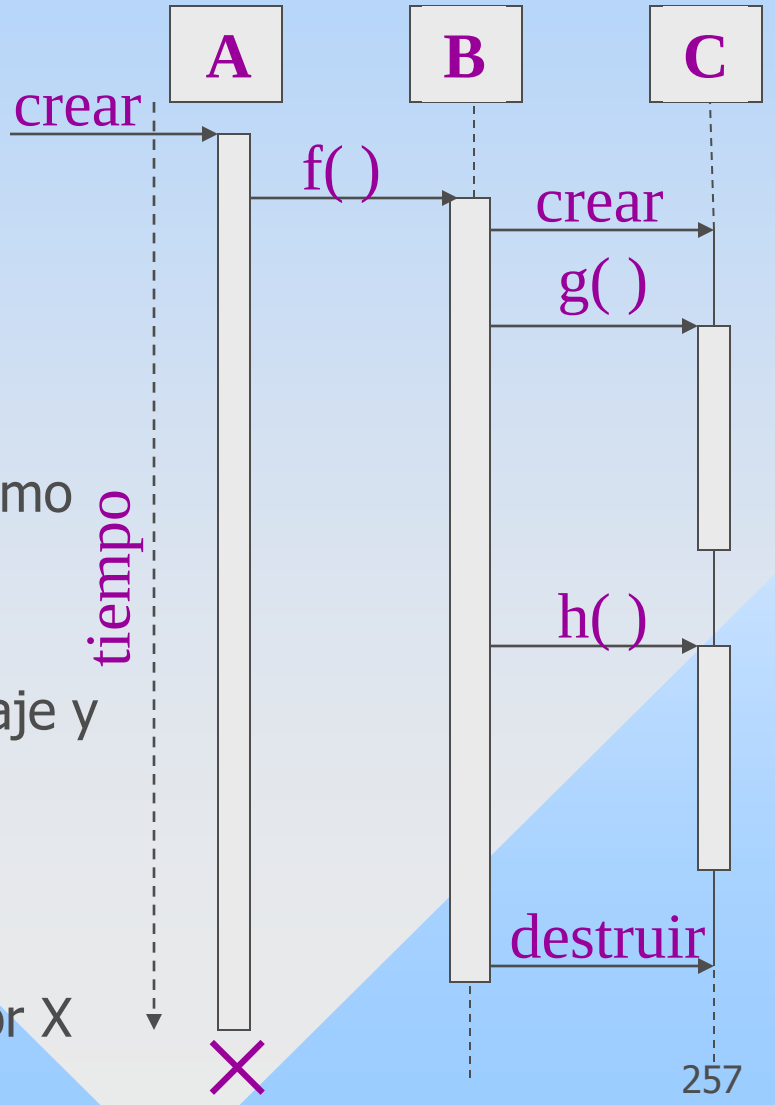
1. Tomar la descripción un caso de uso y convertirla en pseudocódigo colocandola a la derecha del diagrama
2. A partir de la descripción del caso de uso, seleccionar las clases que se tomarán en consideración
3. Para cada uno de los pasos en pseudocódigo, decidir cuales clases tienen la responsabilidad de hacer qué tarea

Receta para la creación de un diagrama de secuencia (2)

4. Para cada una de las tareas, dividirlas en tarea menores
5. Agregar lo correspondiente para los *include* y *extend* del caso de uso
6. Considerar los posibles errores importantes que no fueron cubiertos por el modelo de casos de uso
7. Considerar los nuevos descubrimientos de necesidades y alimentarlos al modelo de casos de uso

Diagrama de secuencias en UML: La notación

- Muestra interacciones entre un conjunto de objetos en forma temporal
- Convenciones de notación:
  - Objetos: Su existencia se muestra como una línea vertical (línea de vida)
  - Mensajes: Líneas horizontales etiquetadas con el nombre del mensaje y sus valores del argumento
  - Tiempo de activación del objeto: Rectángulos delgados
  - Destrucción de objetos: Denotado por X



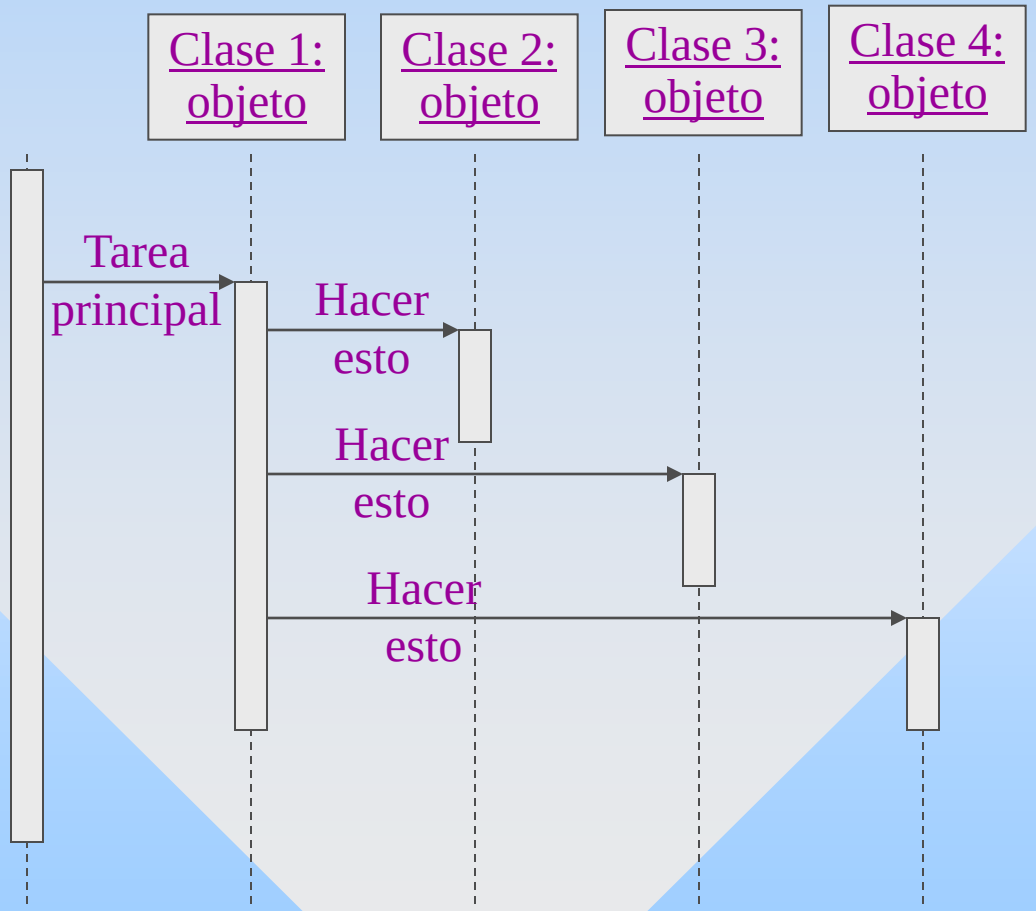
La forma "tenedor" en los diagramas de secuencia

Descripción

Tarea principal
Hacer esto

Hacer esto

Hacer esto

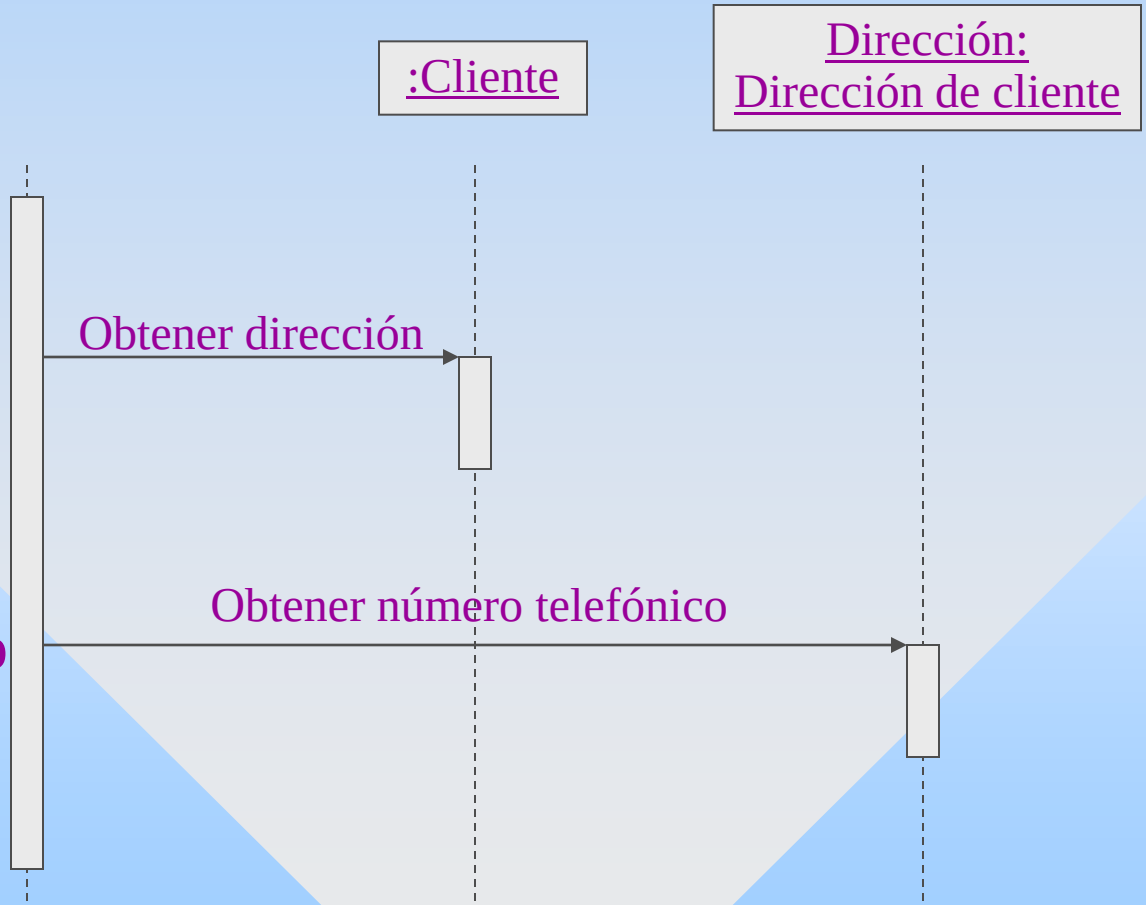


Ejemplo de la forma "tenedor"

Descripción

Obtener dirección

Obtener número telefónico



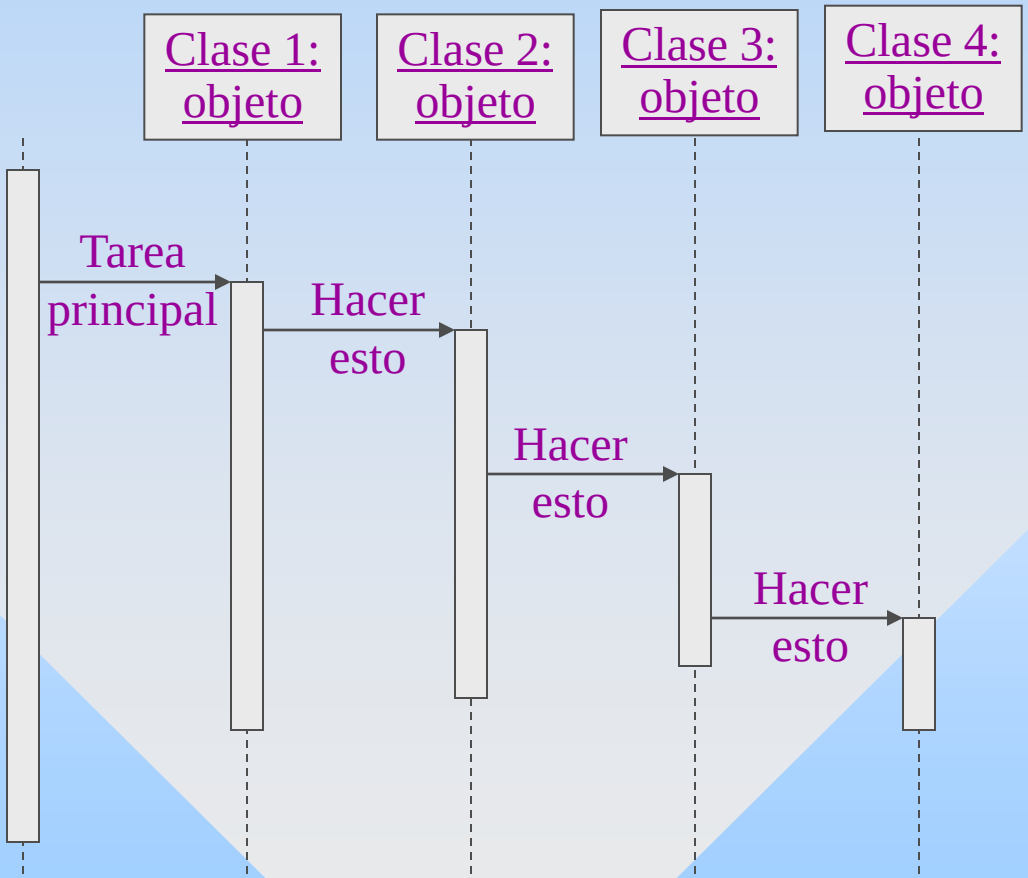
La forma "escalera" en los diagramas de secuencia

Descripción

Tarea principal
Hacer esto

Hacer esto

Hacer esto

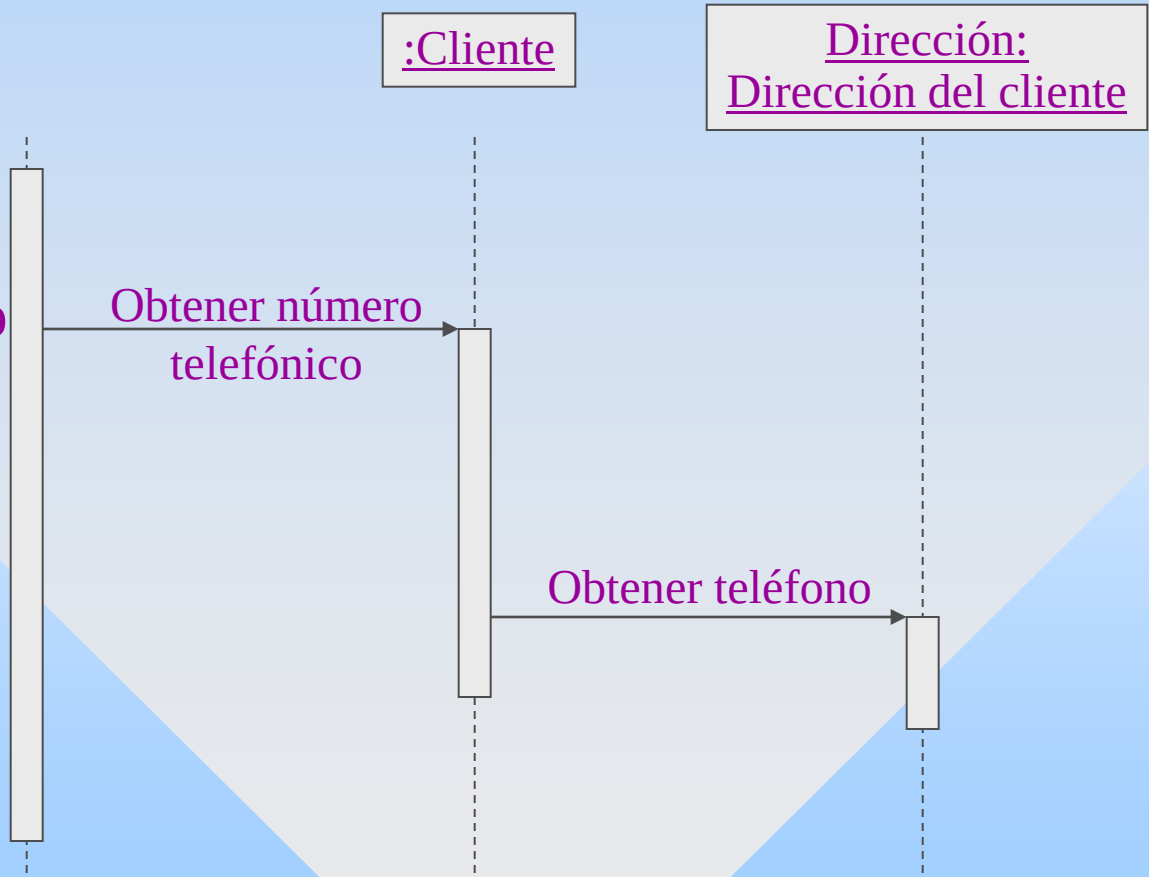


Ejemplo de la forma "escalera"

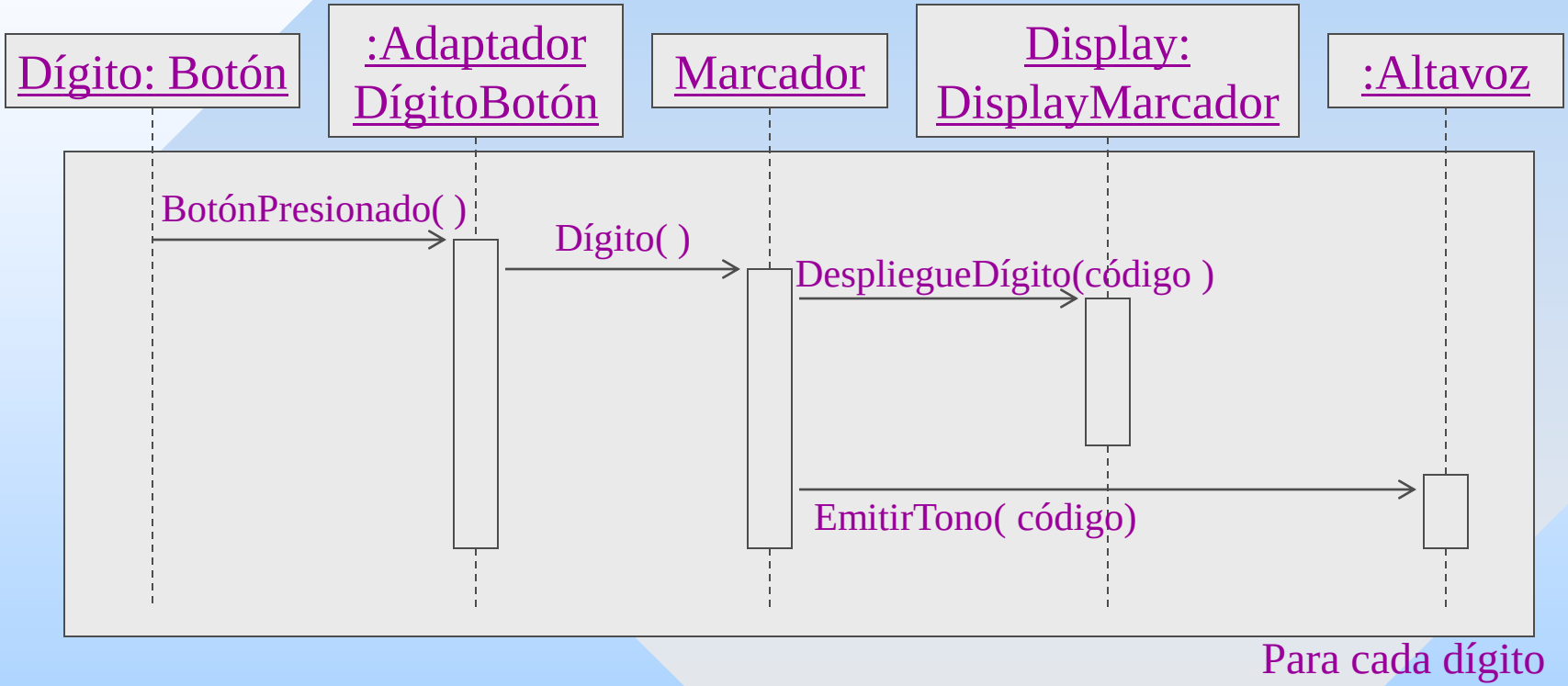
Descripción

Obtener número telefónico

...

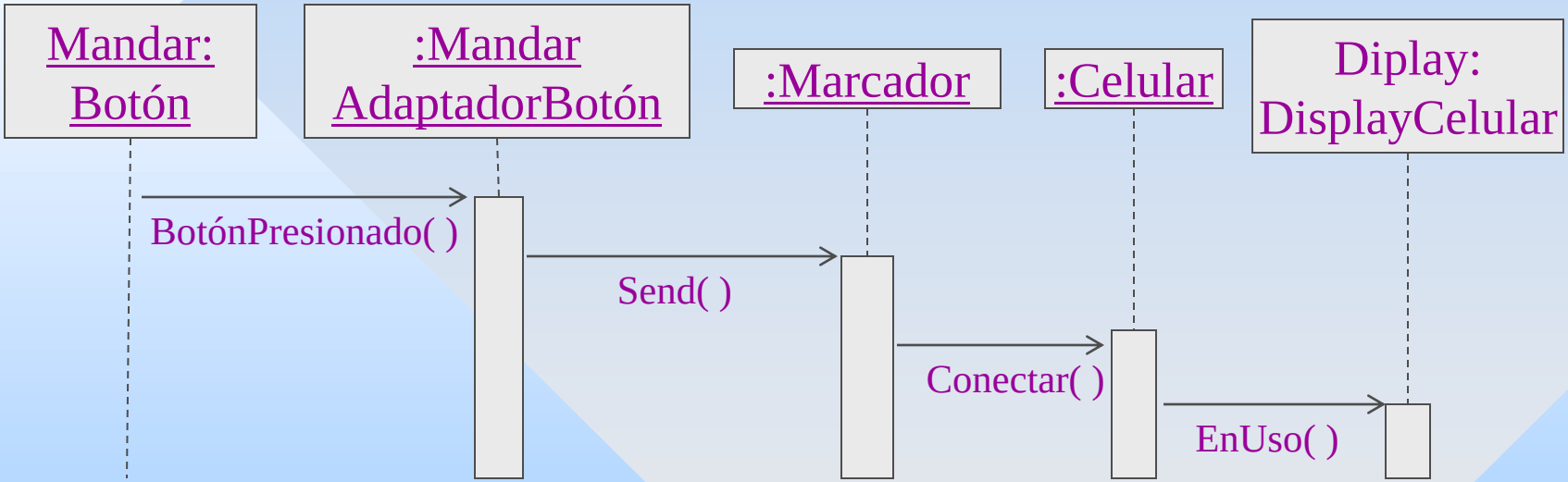


Ejemplo: Teléfono celular (marcado)

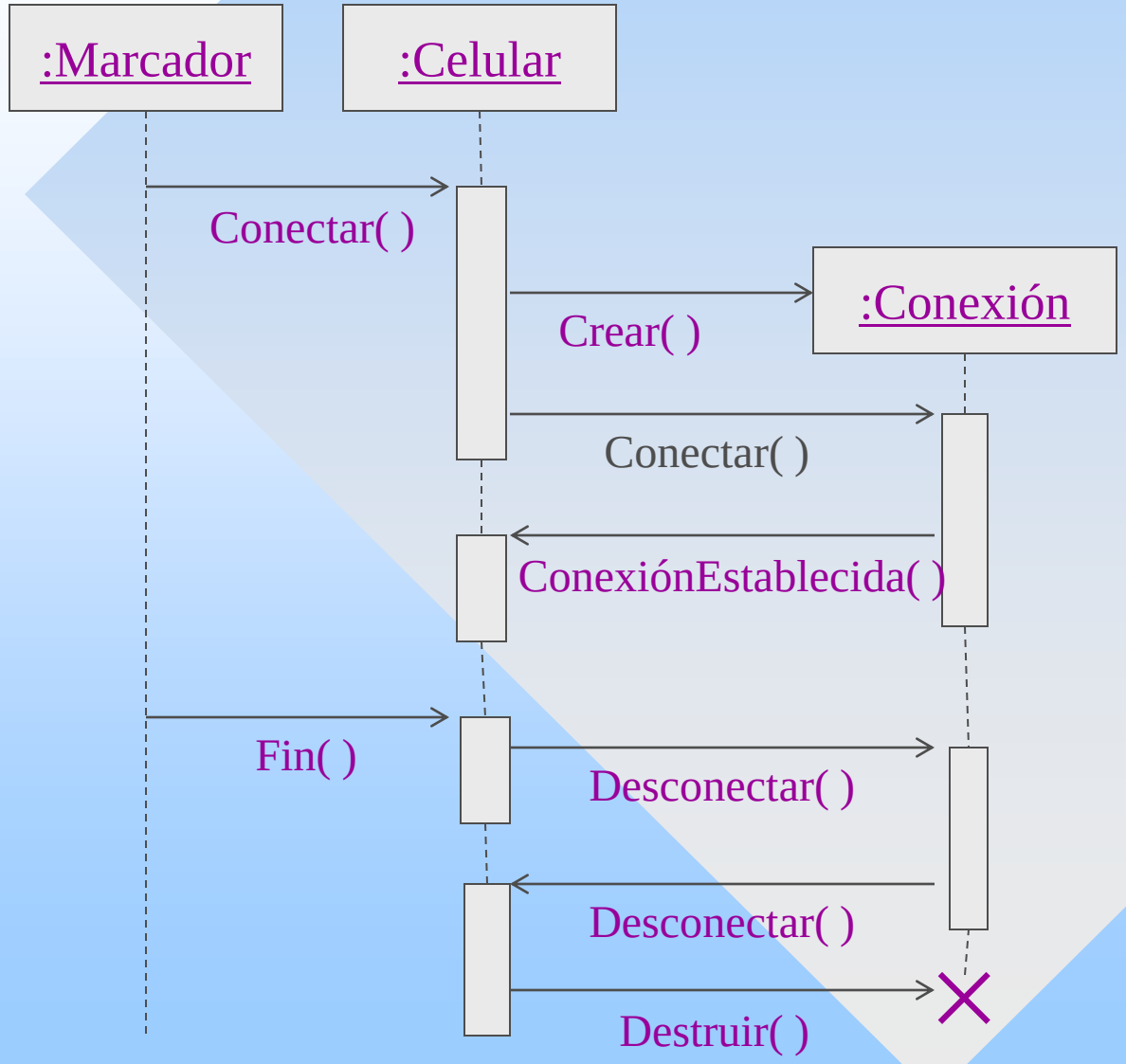


- El rectángulo que encierra al grupo de mensajes define una iteración
  - La condición del ciclo se muestra en la base del rectángulo

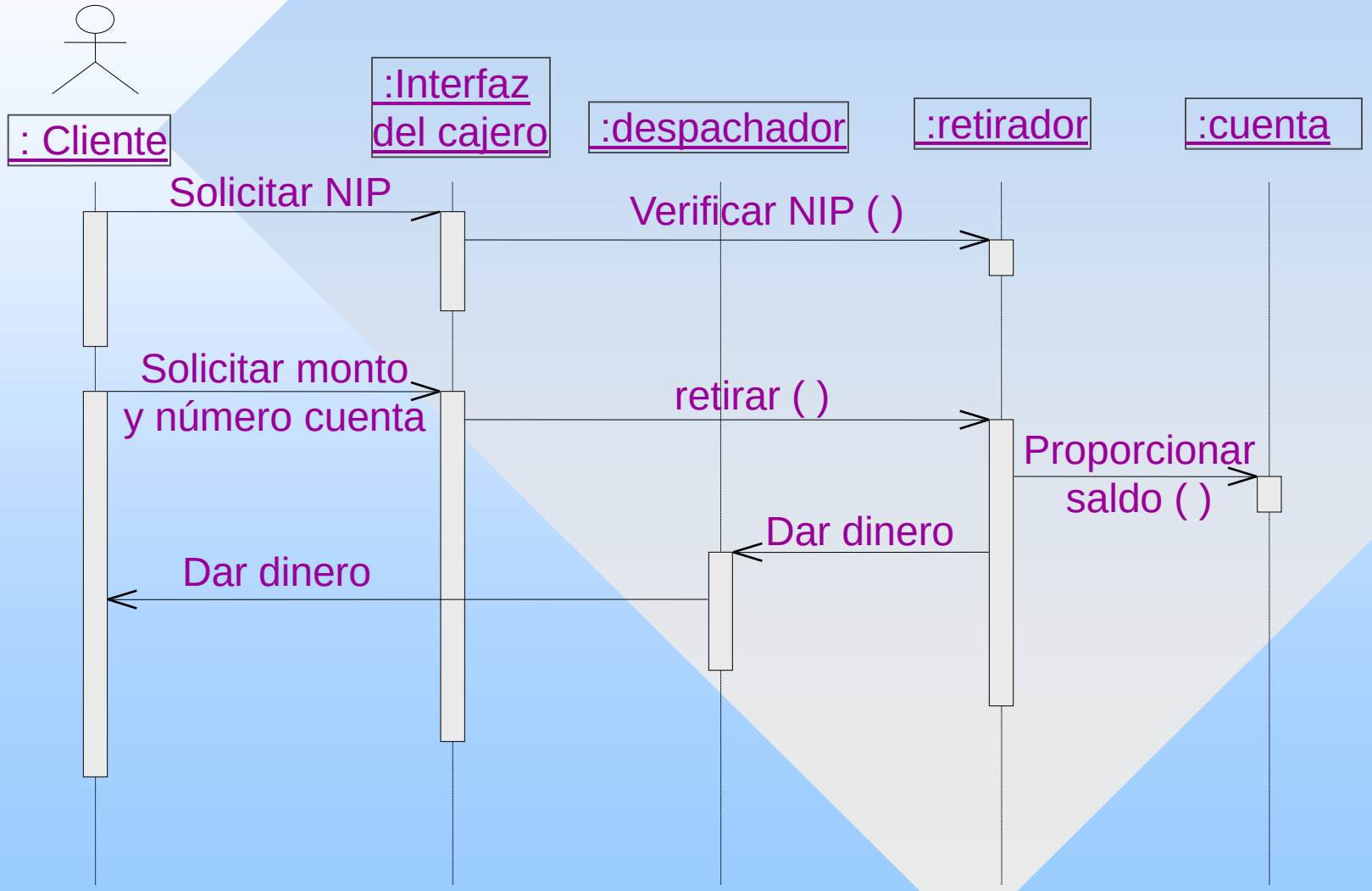
Ejemplo: Teléfono celular ("*send*")



Ejemplo: Teléfono celular (conectar y desconectar)



Ejemplo: Cajero automático



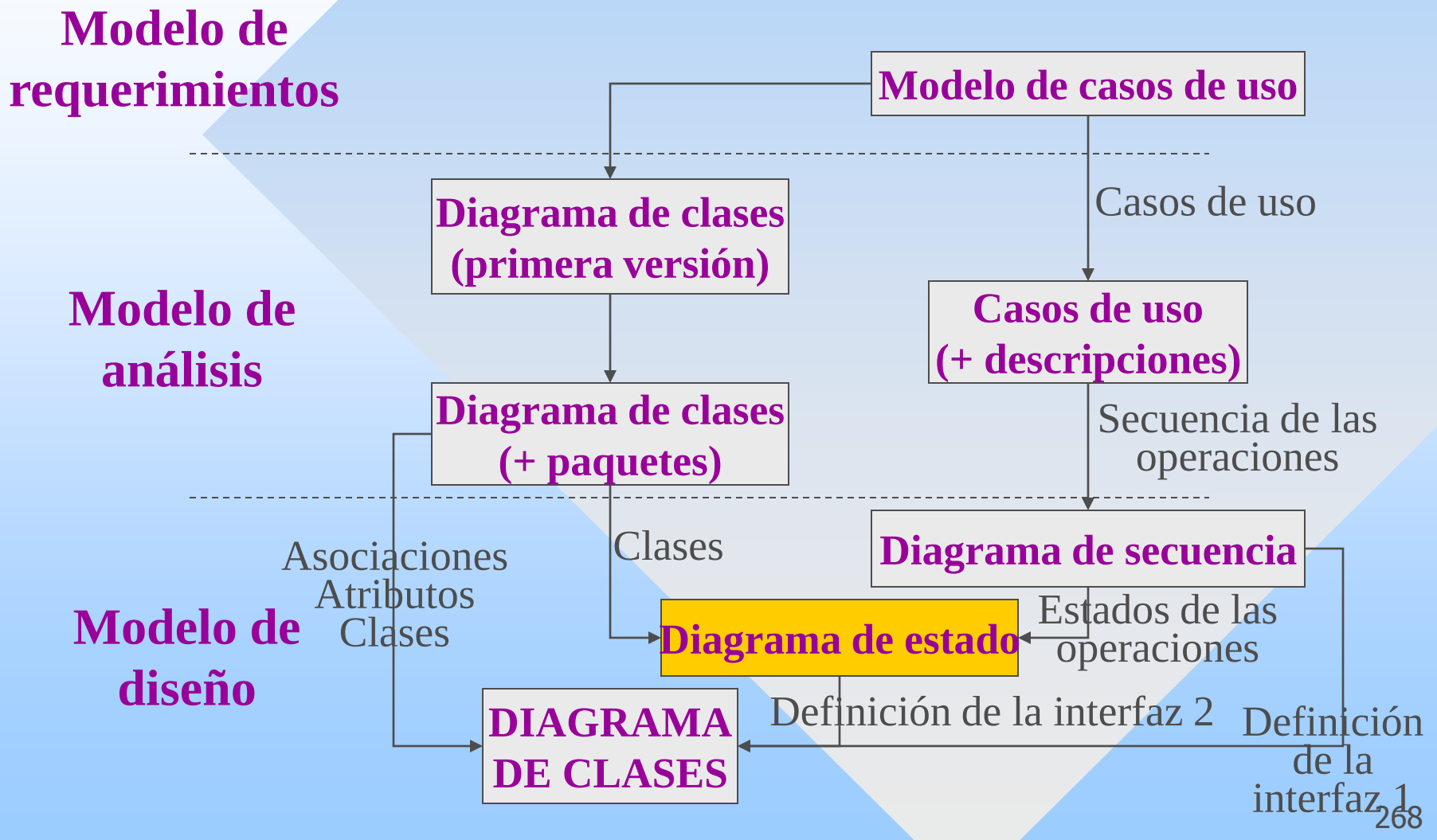
EL MODELO DE DISEÑO (DIAGRAMAS DE ESTADO)

Jorge Kashiwamoto

Panorámica general

- Definir el concepto de Evento, Actividad y Estado
- Comprender las partes de un Estado y de Transición
- Analizar ejemplos de diagramas de Estado
- Comprender distintos tipos de diagramas de Estado

El modelo de diseño I



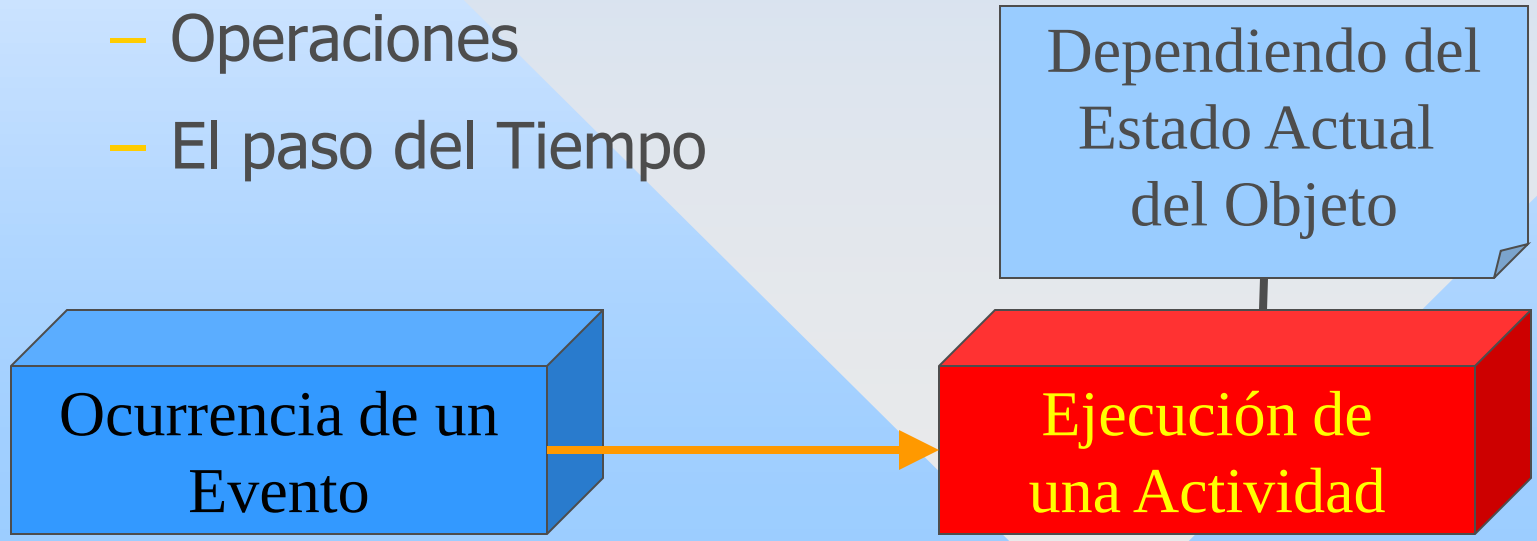
Representación del Comportamiento

- Modelado del aspecto Dinámico
- Interacción
  - Comportamiento de una sociedad de objetos

- Máquina de Estados
  - Comportamiento de un objeto

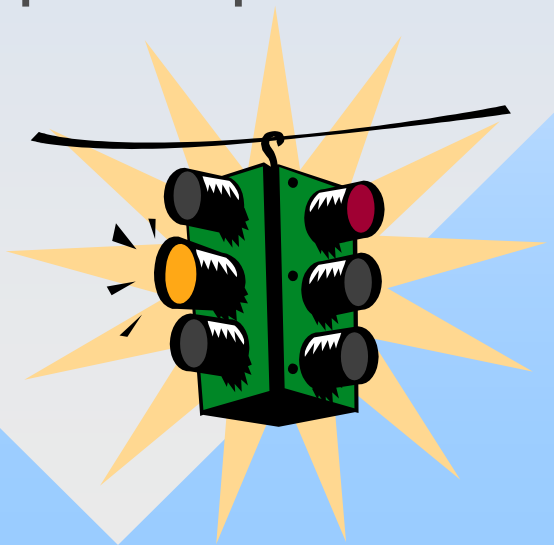
Máquina de Estados

- Secuencia de estados del tiempo de vida de un objeto, suscitados por Eventos
  - Señales
  - Operaciones
  - El paso del Tiempo



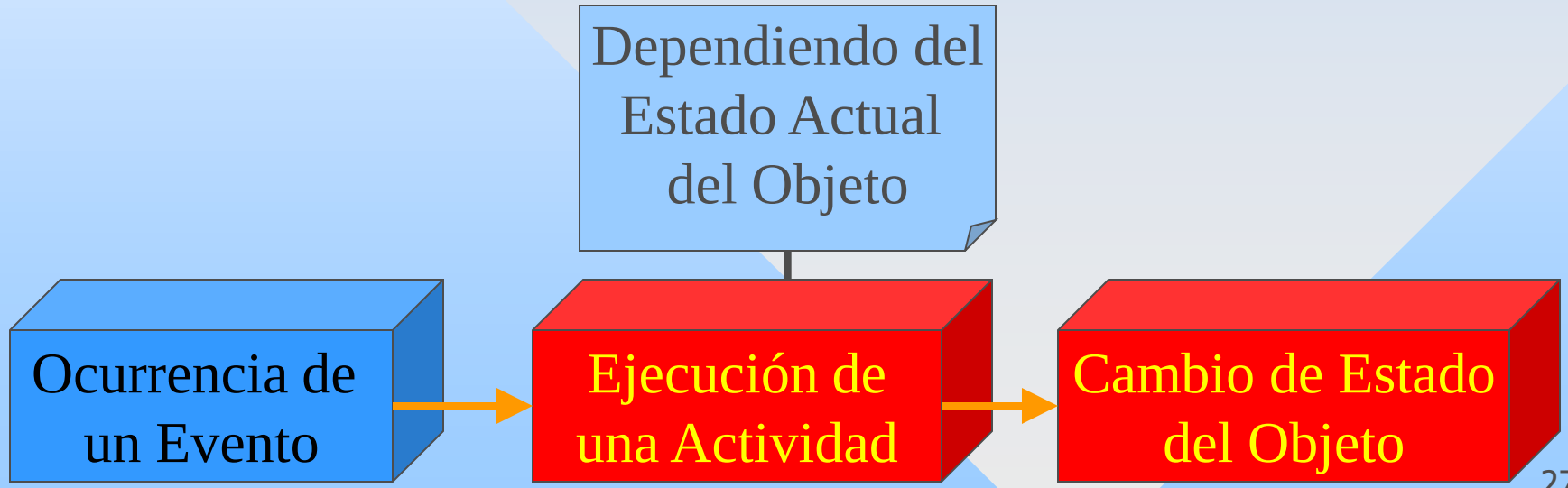
Evento

- Especificación de una ocurrencia significativa que tiene una ubicación en el tiempo y el espacio
- Ocurrencia de un estímulo que dispara la transición de un estado



Actividad

- Ejecución no-atómica dentro de una maquina de estados
- Produce
  - Cambio de Estado
  - Regresa un valor

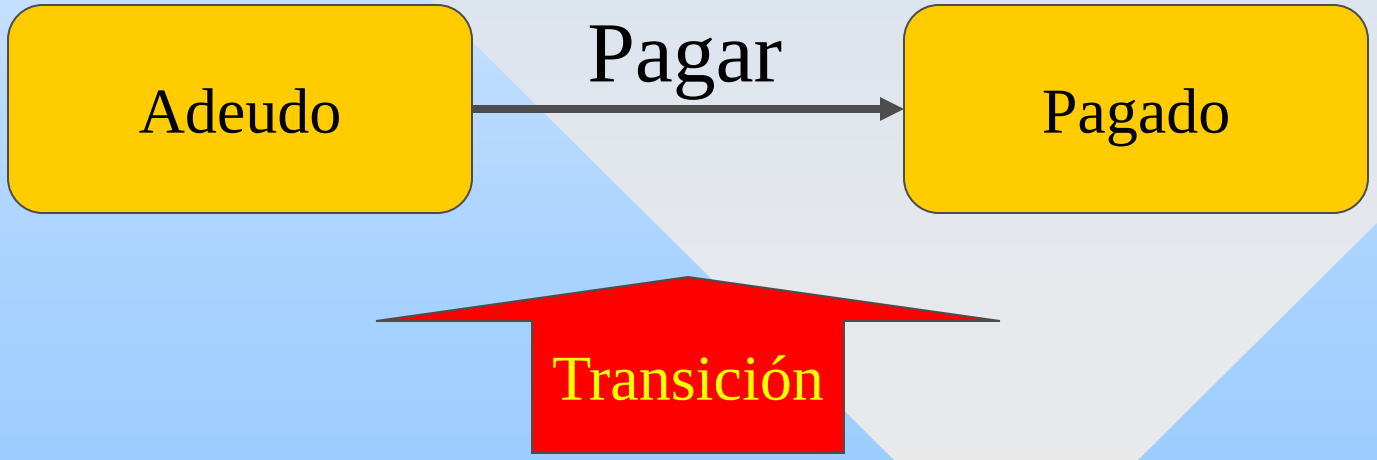


Estado

- Condición o situación de un objeto durante su tiempo de vida, en la cual
  - Satisface una condición
  - Realiza alguna actividad
  - Espera la ocurrencia de un evento
- La respuesta es afectada por el pasado

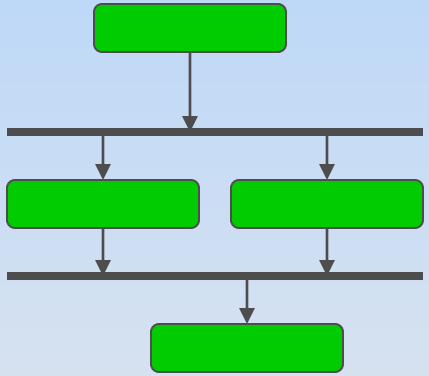
Transición

- Relación de cambio entre 2 estados

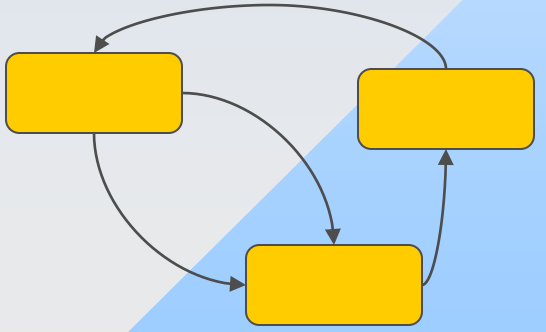


Visualización de una Máquina de Estados

- Por el Flujo de Control
  - DIAGRAMA DE ACTIVIDAD

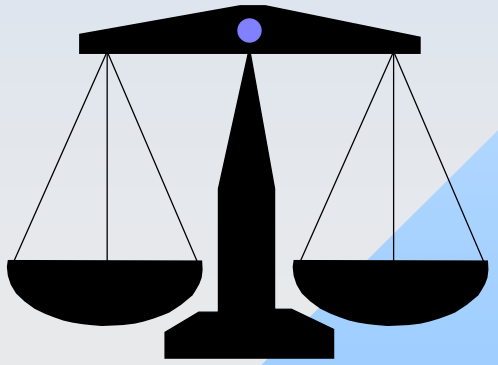


- Por los Estados Potenciales y sus Transiciones
  - DIAGRAMA DE ESTADOS



Características de una Máquina de Estados

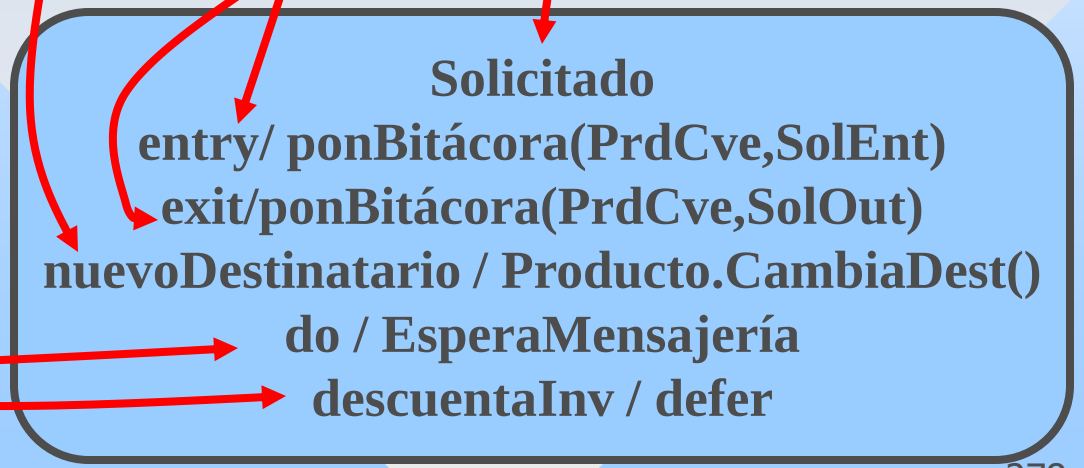
- Una máquina de estados bien estructurada es:
  - Eficiente
  - Simple
  - Adaptable
  - Entendible



Representación de un Estado

Partes de un Estado

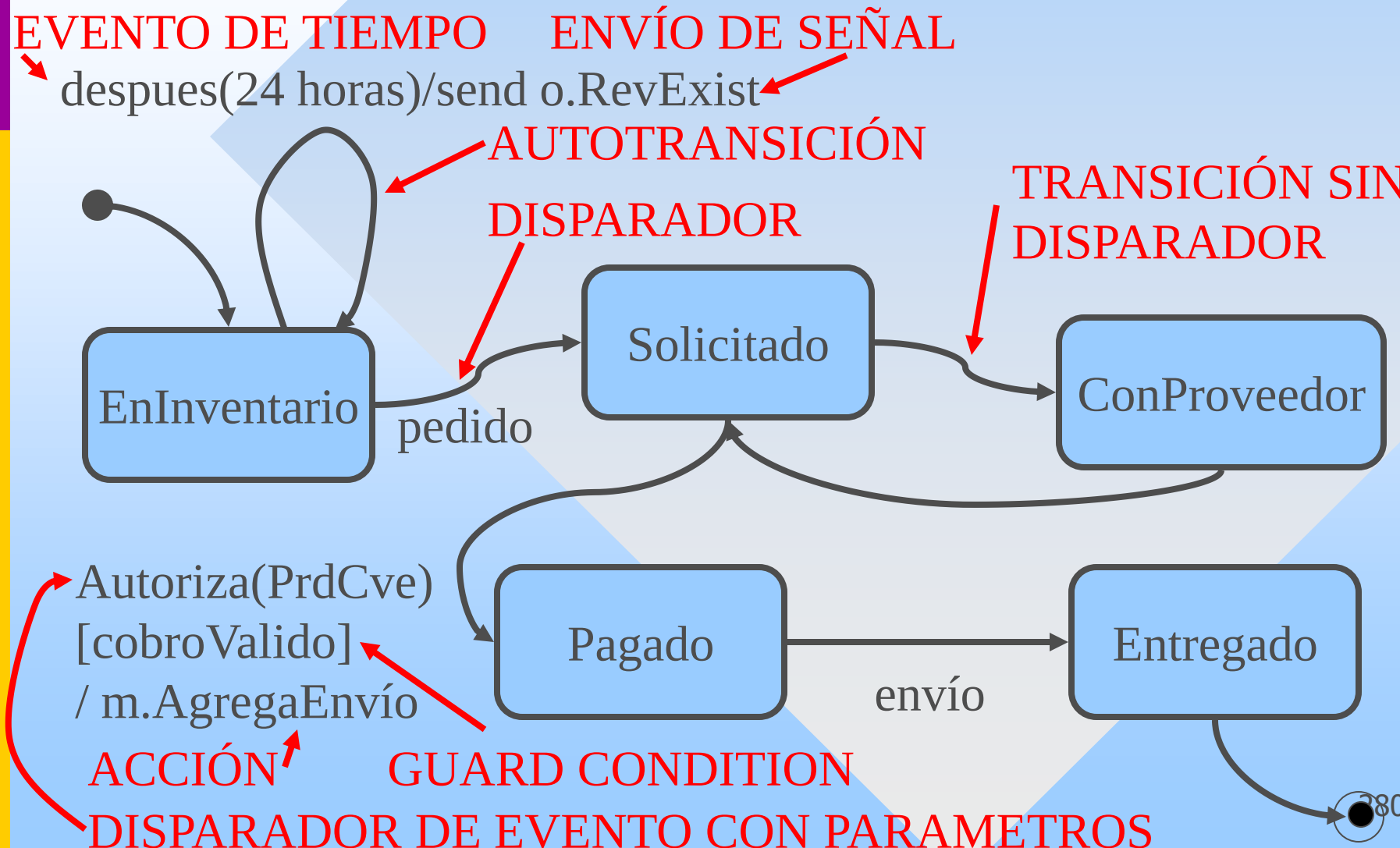
- Nombre
- Acciones de Entrada/Salida
- Transiciones internas
- Subestados
- Eventos Diferidos
- Actividad



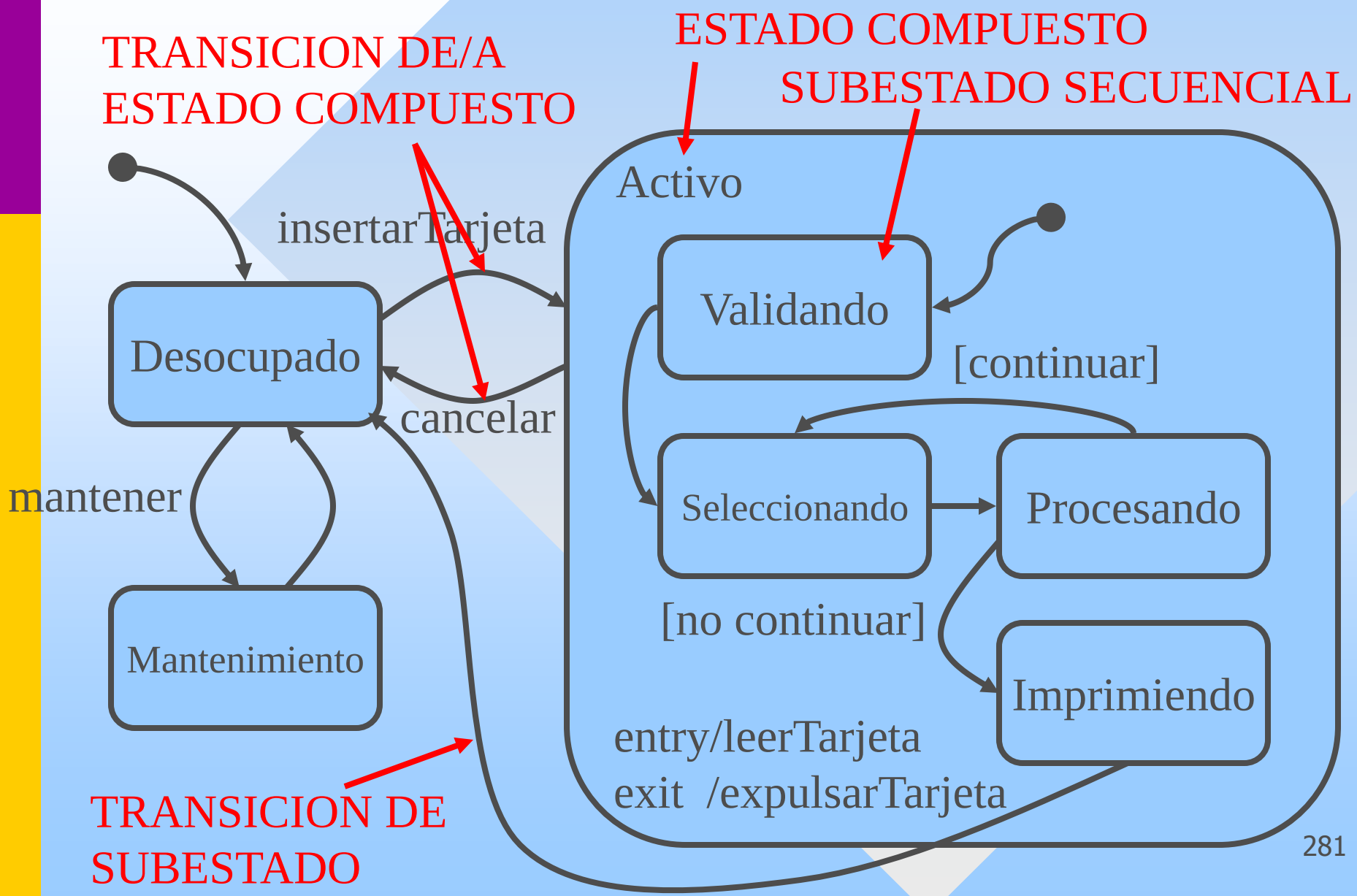
Partes de una Transición

- Estado Origen
- Disparador de evento
  - Event Trigger
- Condición de transición
  - Guard Condition
- Acción
- Estado Destino

Partes de una transición

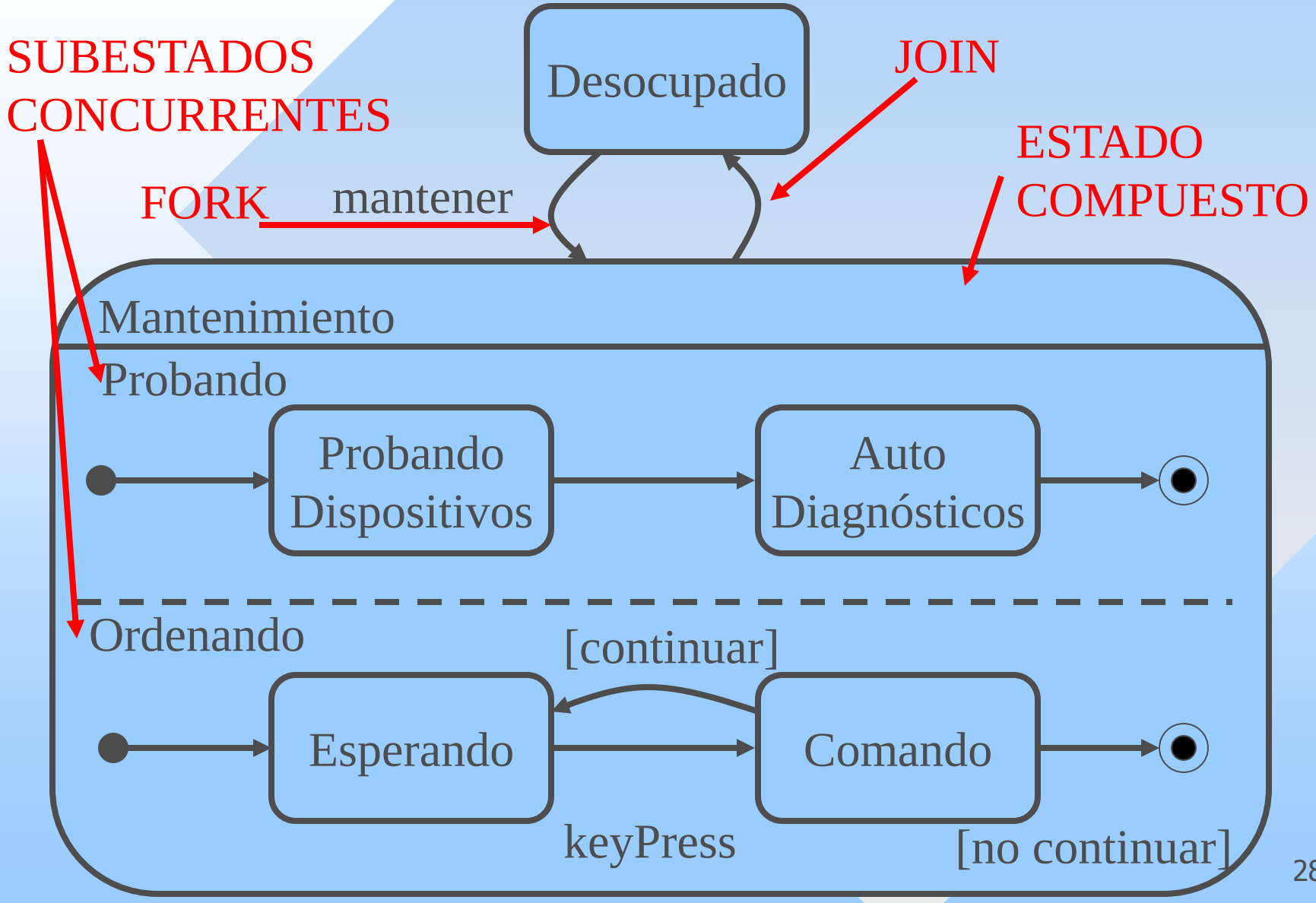


Subestados secuenciales



Subestados históricos

Subestados concurrentes



EL MODELO DE DISEÑO (DIAGRAMAS DE COLABORACIÓN)

Jorge Kashiwamoto

Panorámica general

- Comprender la utilización de los diagramas de colaboración:
  - Fundamentos
  - Relación con Diagramas de Secuencia
  - Elementos
  - Flujos de Control
  - Usos
  - Pasos de la Aplicación Metodológica

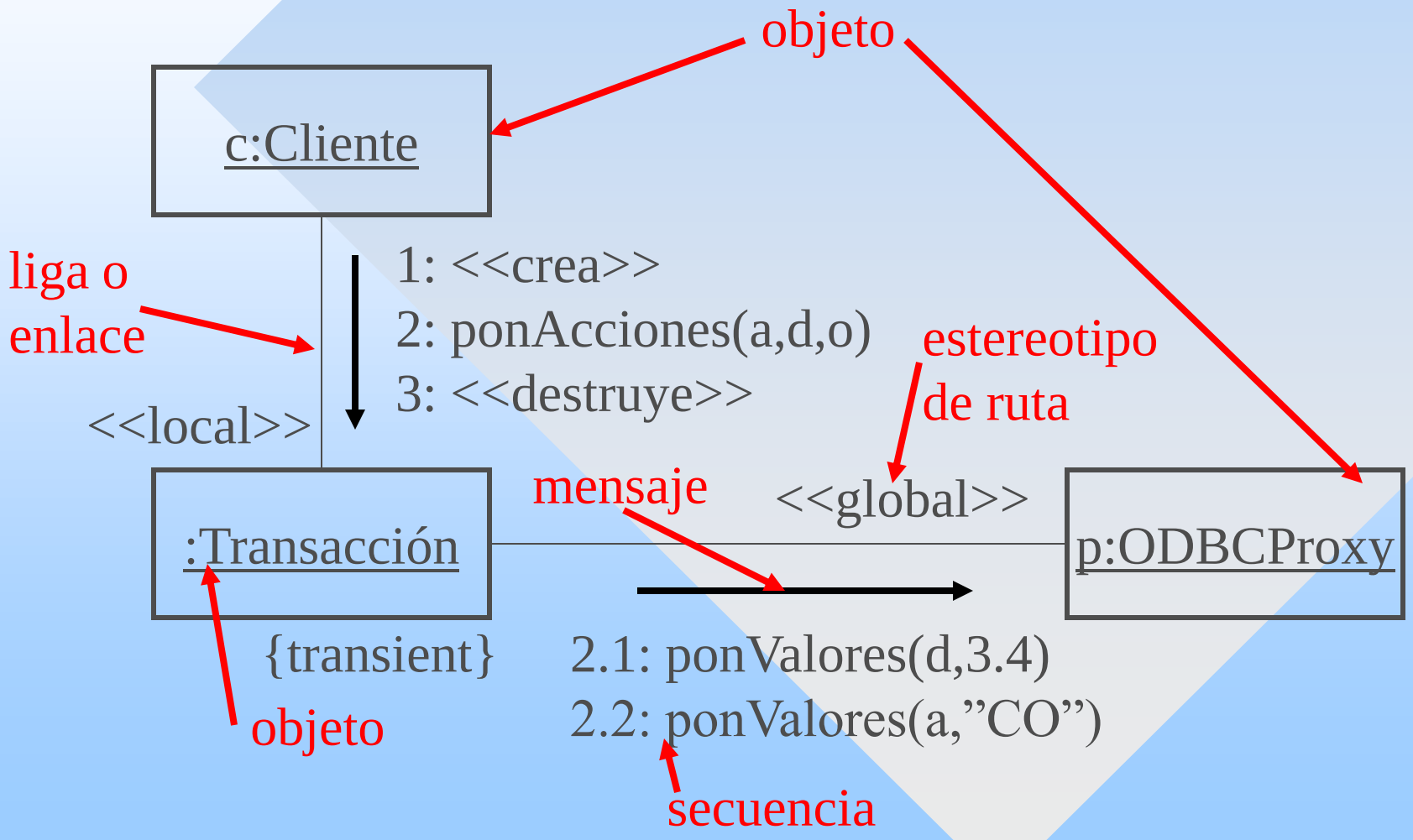
Diagrama de colaboración

- Tipo de diagrama de interacción
- Modela el aspecto dinámico
- Interacción = Conjunto de objetos y sus relaciones (mensajes)
- Enfatiza la Organización Estructural de los objetos que mandan y reciben mensajes

Elementos de un diagrama de colaboración

- Objetos
- Ligas
- Mensajes

Elementos


objeto
c:Cliente
liga o
enlace
1: <<crea>>
2: ponAcciones(a,d,o)
3: <<destruye>>
estereotipo
de ruta
<<local>>
mensaje
<<global>>
:Transacción
p:ODBCProxy
{transient}
2.1: ponValores(d,3.4)
2.2: ponValores(a,"CO")
objeto
secuencia

Diferencias con el diagrama de secuencia

- Primera
  - RUTA
    - Cómo un objeto se liga con otro
      - <<local>> el objeto es local al objeto cliente
      - <<global>>
      - <<parameter>>
      - <<self>>

Diferencias con el diagrama de secuencia

- Segunda
  - NUMERACIÓN DE LA SECUENCIA
    - Indicación del Orden de Ejecución
    - Se precede cada mensaje con un número
    - Numeración decimal Dewey para subsecuencias

Flujos

- Flujo Secuencial de Control
- Flujos Complejos
  - Iteración
    - Precediendo con
      - *
      - *[i := 1..n]
  - Salto
    - [condición_booleana]
    - Ramas con la misma numeración
  - Las expresiones pueden ser pseudocódigo o de algún lenguaje en particular.

Equivalencia semántica

- Los diagramas de colaboración y de secuencia tienen equivalencia semántica.
- Aunque no necesariamente explícitamente visualicen la misma información.

¿Cuándo usar diagramas de colaboración?

- Enfatizar las relaciones estructurales entre las instancias de la interacción (mensajes)
- Visualizar iteraciones y saltos complejos
- Visualizar múltiples flujos concurrentes de control

Pasos

- 1. Establecer el contexto
  - (sistema, subsistema, operación, clase, escenario de caso de uso, colaboración)
- 2. Identificar los objetos.
  - Colocarlos como vértices
  - Los objetos más importantes van al centro

Pasos

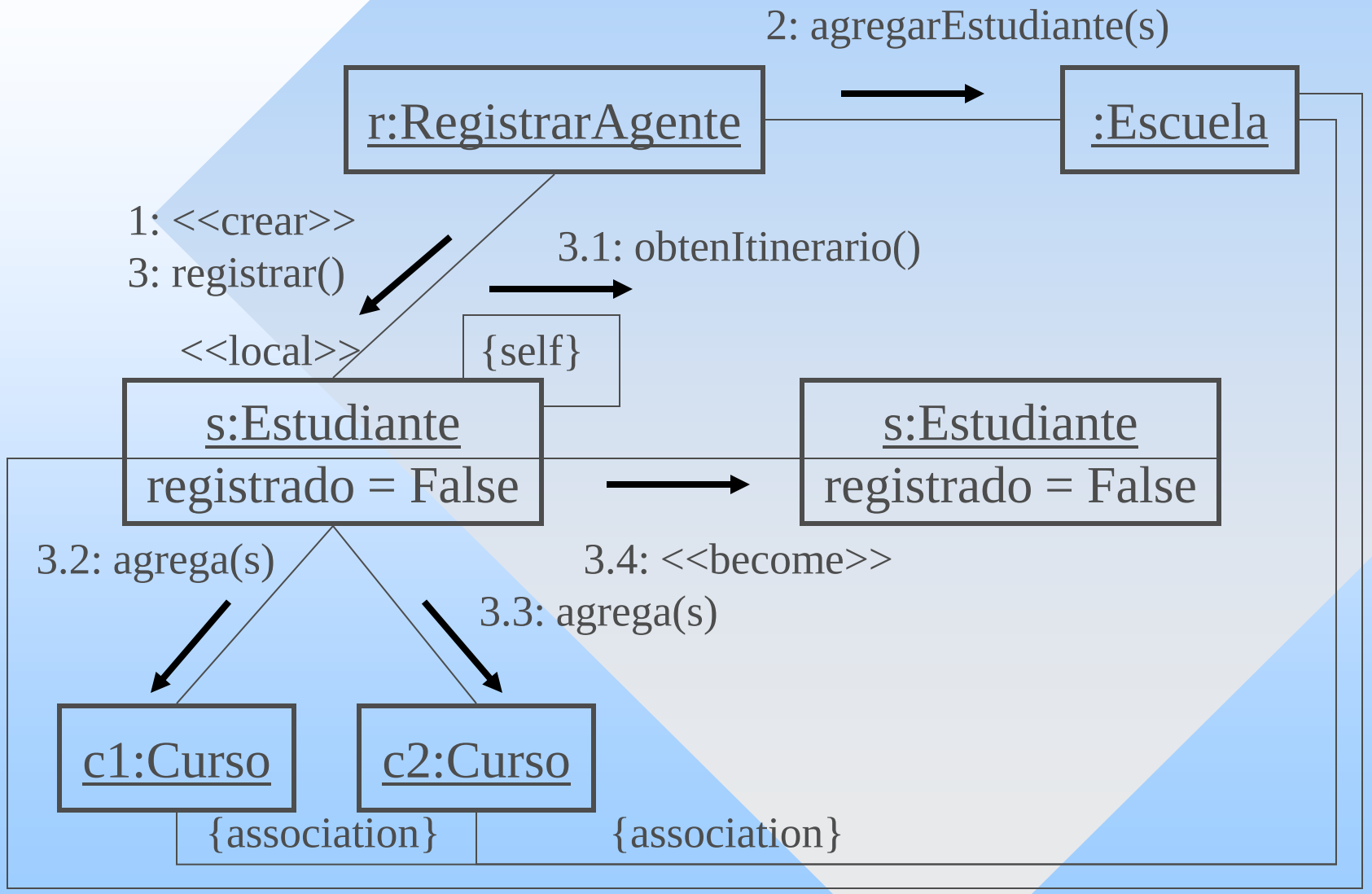
- 3. Establecer los valores iniciales de cada uno de estos objetos
  - (valores, estados, roles, instanciación múltiple –become, copy -)
- 4. Especificar la liga entre los objetos y los mensajes a ser utilizados
  - 4.1 Trazar las ligas de asociación: Conexiones estructurales
  - 4.2 Trazar otras ligas y catalogarlas con Estereotipos de Ruta (global, local)

Pasos

- 5. Iniciar el trazo de mensajes
  - 5.1 Comenzar con el mensaje que inicia la interacción
  - 5.2 Añadir mensajes subsecuentes
  - 5.3 Aplicar la secuencia de numeración
- 6. Si es necesario, aplicar restricciones de tiempo o espacio.
- 7. Un control más formal implica precondiciones y postcondciones para cada mensaje.

Otros consejos

- Usualmente los diagramas de secuencia muestran un flujo de control
- Típicamente, se tienen varios diagramas de interacción
  - Primarios
  - Rutas alternativas


2: agregarEstudiante(s)
r:RegistrarAgente
:Escuela
1: <<crear>>
3: registrar()
3.1: obtenItinerario()
<<local>>
{self}
s:Estudiante
registrado = False
s:Estudiante
registrado = False
3.2: agrega(s)
3.4: <<become>>
3.3: agrega(s)
c1:Curso
c2:Curso
{association}
{association}

Diagramas de Interacción Bien Estructurados

- Se enfoca en un aspecto de la dinámica del sistema
- Contiene solamente aquellos elementos que son esenciales para entender un aspecto del sistema
- Provee detalles consistentes con su nivel de abstraccón y aquellos adornments esenciales
- No es demasiado general ni exageradamente detallado

EL MODELO DE DISEÑO (DIAGRAMAS DE ACTIVIDAD)

Jorge Kashiwamoto

Panorámica general

- Comprender la utilización de los diagramas de Actividad:
  - Fundamentos
  - Relación con Diagramas de Secuencia
  - Elementos
  - Flujos de Control
  - Usos
  - Pasos de la Aplicación Metodológica

Diagrama de actividad

- Muestra el Flujo de Actividad a Actividad
- Modela el aspecto dinámico de una sociedad de objetos
- Modelado de pasos
  - Secuencia (de un procedimiento computacional)
  - Concurrente

Diagrama de actividad (2)

- Modela el Flujo de un Objeto, de cómo se mueve de un estado a otro en los diferentes flujos de control.
- Comprando con D. de Interacción:
  - D.Interacción: Flujo de control de objeto a objeto.
  - D.Actividad: Flujo de control de actividad a actividad.

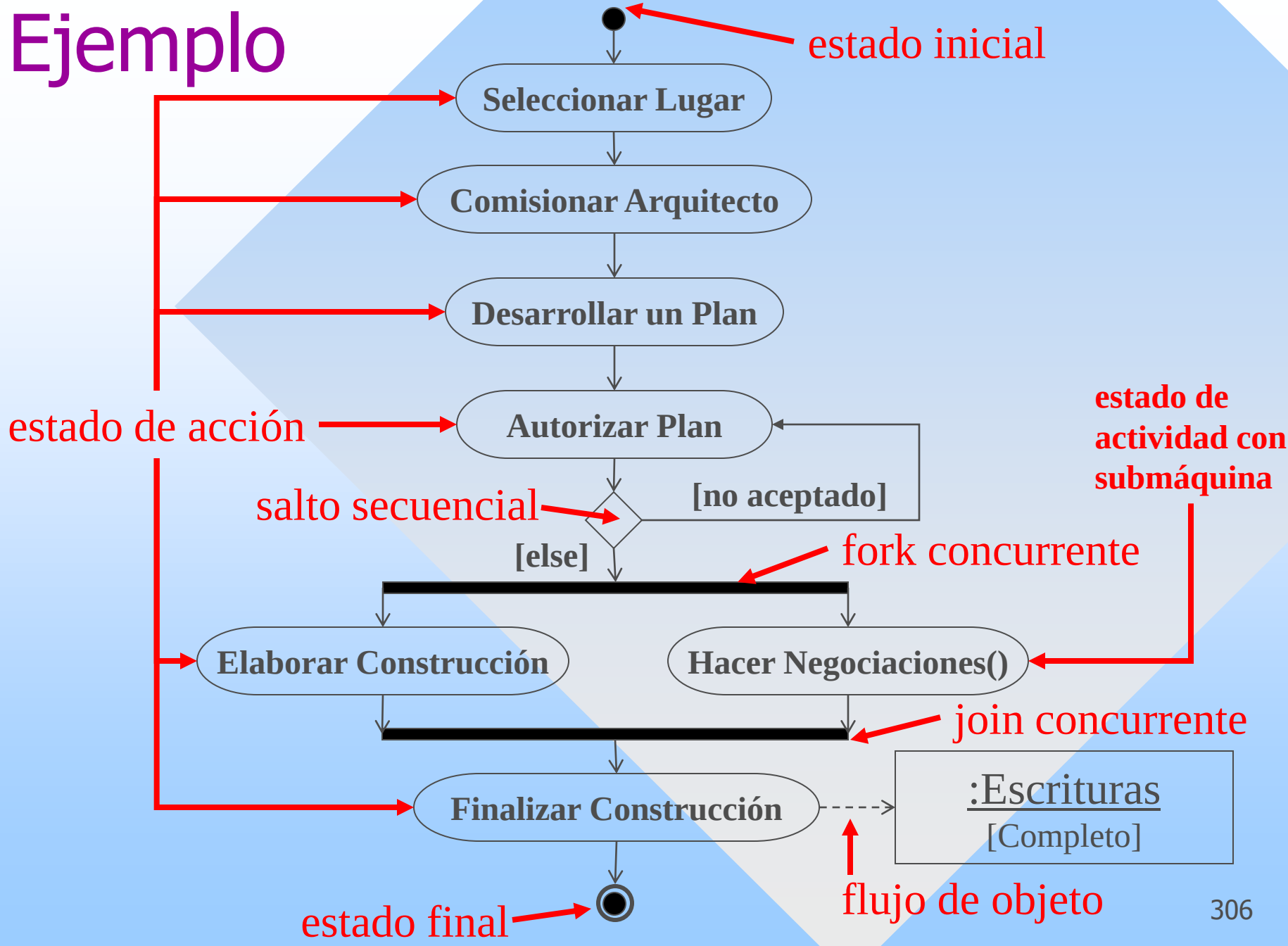
Actividad

- Ejecución no atómica dentro de una máquina de estados.
- Las actividades resultan en una Acción (cálculos atómicos ejecutables):
  - Cambio de Estado
  - Regreso de un Valor

Acciones

- Agrupan a
  - La llamada a otra operación
  - Enviar una señal
  - Crear o destruir objetos
  - Cálculos puros
    - V.g.: Evaluar una expresión

Ejemplo


estado inicial
Seleccionar Lugar
Comisionar Arquitecto
Desarrollar un Plan
Autorizar Plan
estado de acción
[no aceptado]
salto secuencial
[else]
fork concurrente
estado de actividad con submáquina
Elaborar Construcción
Hacer Negociaciones()
join concurrente
Finalizar Construcción
:Escrituras
[Completo]
flujo de objeto
estado final

Estados Acción

- Representan una acción
- No pueden descomponerse, es atómico
- No puede interrumpirse
- Tiempo de ejecución insignificante

Estados Acción

expresión

Autorizar Plan

acción sencilla

i := Sqrt(j) + 1

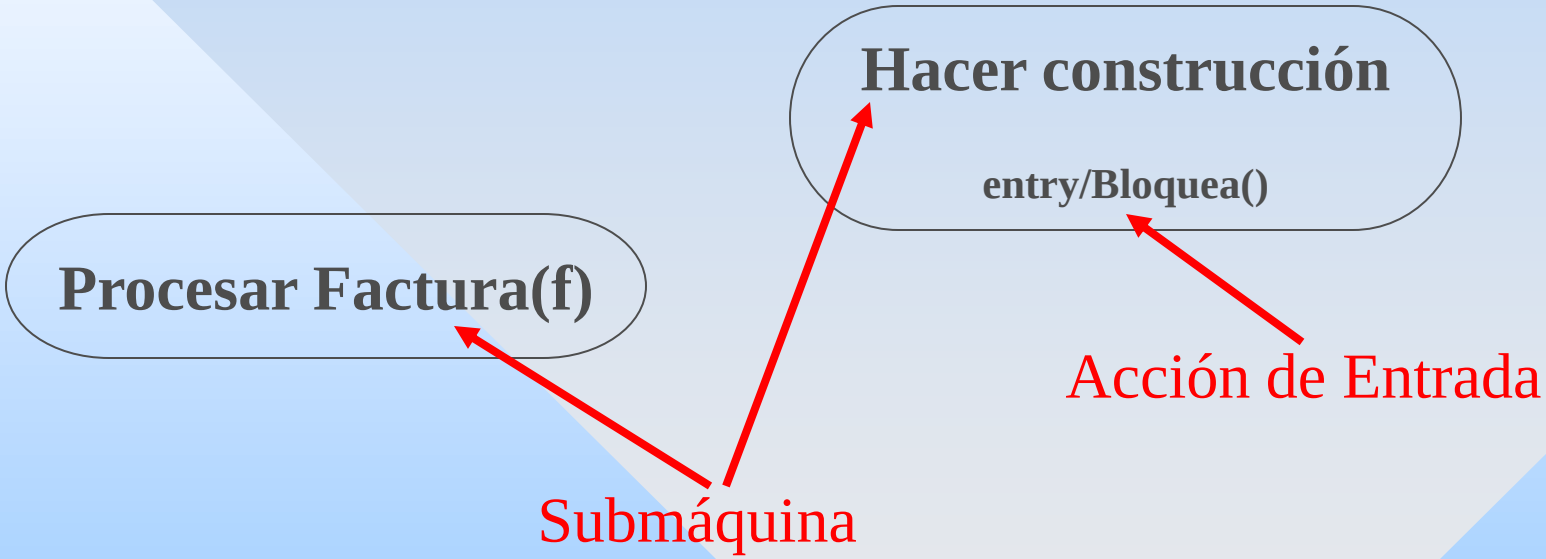
Estados Actividad

- Pueden descomponerse
- Su actividad está respresentada por otros diagramas de actividad
- No son atómicos, pueden interrumpirse
- Se considera que toma algún tiempo en ejecutarse
- Un Estado Acción es un subcaso del Estado Actividad

Estados Actividad

- Estado Actividad =
  - + Estados Acción
  - + Estados Actividad
- Partes
  - Acción de Entrada
  - Acción de Salida
  - Especificaciones de Submáquina

Estados Actividad



Transiciones

- Es el paso de un estado acción o actividad a otro, una vez que éste se ha completado.
- Se representan con líneas con puntas de flecha dirigidas
- Se denominan de tipo ‘sin disparador’ o ‘triggerless’.

Saltos

- Especificación de rutas alternativas basadas en expresiones Booleanas.
- Se representa con un rombo a partir del cual, se tienen más de una transición alternativa especificadas con Guard Conditions entre corchetes.
- Se usa la palabra reservada ‘else’
- Pueden modelarse iteraciones.

Fork / Join

- Modelar actividades concurrentes
- Pueden anidarse
- Se representan con líneas horizontales a partir de las cuales salen las transiciones iniciales o llegan las finales de los estados concurrentes.

Swimlanes

- Grupos o particiones de estados de actividad según la organización de negocio responsable.
- Modelar flujos de trabajo de Procesos de Negocio.
- Se representan con rectángulos etiquetados con la unidad organizacional dentro de los cuales se agrupan los estados.

Flujos de Objeto

- Modela los objetos (particulamente con estado) que pasan de un estado a otro.
- Se representa como se hace convencionalmente en UML y se une con líneas rayadas dirigidas entre los estados.

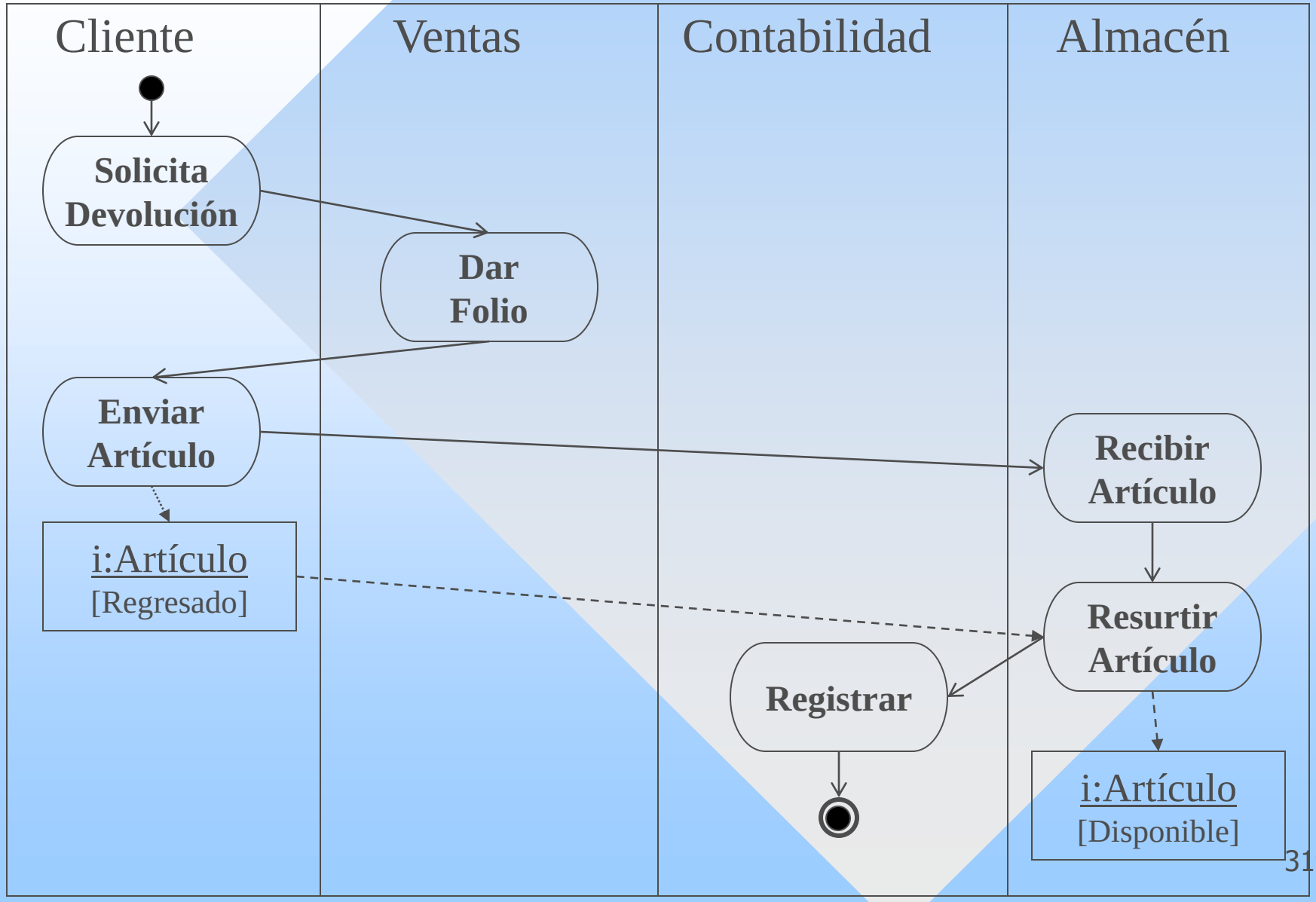
Aplicación Metodológica para Flujos de Trabajo

- 1. Establecer el foco del flujo de trabajo
- 2. Seleccionar los objetos de negocio de alta nivel de responsabilidad
- 3. Identificar las precondiciones para el estado inicial y las postcondiciones para el estado final
- 4. Se trazan los elementos

Flujos de Trabajo

- 5. Para acciones o conjunto de acciones complicadas, se colapsan en estados Actividad.
- 6. Trazar las transiciones, comenzar con las secuenciales, saltos y concurrentes en este orden.
- 7. Se grafican los objetos importantes con sus valores o estados.

Ejemplo


Cliente
Ventas
Contabilidad
Almacén
Solicita Devolución
Dar Folio
Enviar Artículo
i:Artículo
[Regresado]
Recibir Artículo
Resurtir Artículo
Registrar
i:Artículo
[Disponible]

Aplicación Metodológica para Operaciones

- 1. Se agrupan las abstracciones involucradas en la operación (parámetros, atributos y clases)
- 2. Identificar precondiciones al estado inicial y postcondiciones al estado final.
- 3. Comenzar con el trazado de estados.
- 4. Especificar saldos y concurrencia.

EL MODELO DE IMPLANTACIÓN (DIAGRAMAS DE COMPONENTES)

Jorge Kashiwamoto

Panorámica general

- Comprender la utilización de los diagramas de componentes:
  - Fundamentos
  - Relación con Clases
  - Relación con Interfaces
  - Reemplazo Binario
  - Tipos
  - Ejemplos

Diagrama de Componentes

- Modelado de Aspectos Físicos del Sistema
- Muestra de un conjunto de componentes
  - la Organización, y
  - la Dependencia
- Modela una vista de implantación estática de un sistema

Diagrama de Componentes

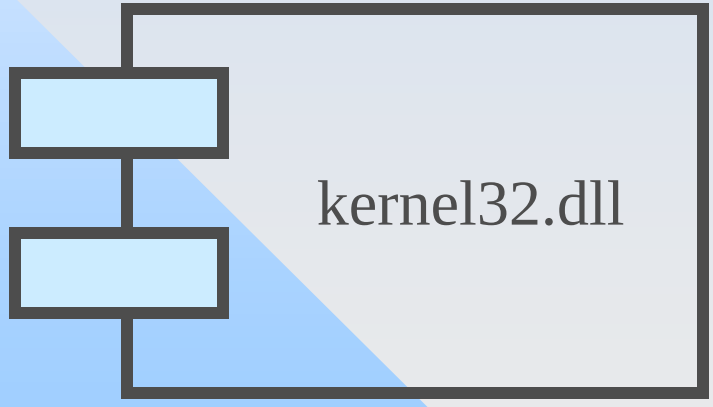
- Involucra el modelado de cosas físicas que residen en un NODO
  - Ejecutables
  - Bibliotecas
  - Tablas
  - Archivos
  - Documentos

Diagrama de Componentes

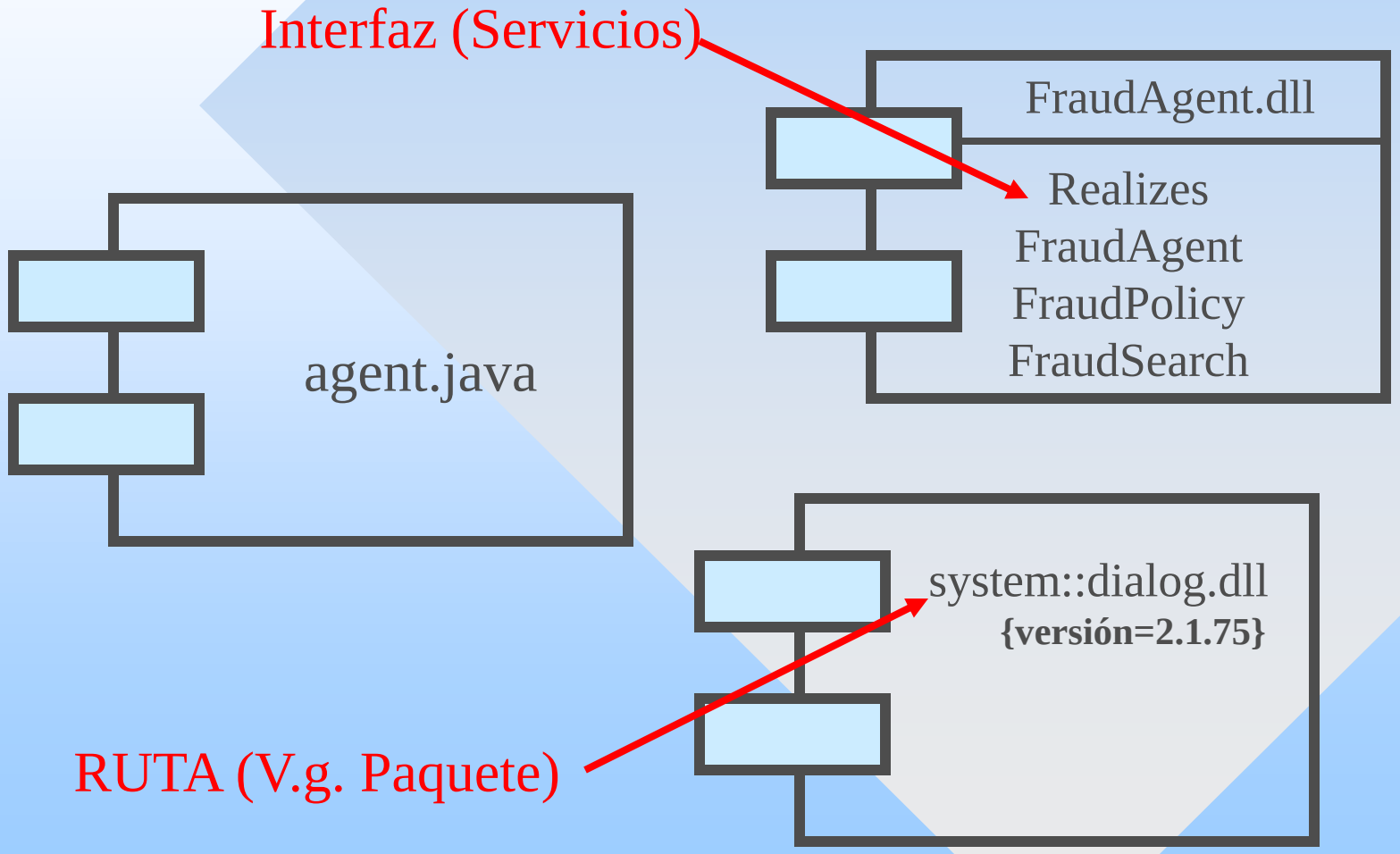
- Definición
  - Visualiza
    - Un conjunto de componentes y
    - Sus relaciones
  - Gráficamente es un conjunto de
    - Vértices
    - Arcos

Componente

- Parte física y reemplazable de un sistema que conforma la realización de un conjunto de interfaces



Nombre de un componente



Componentes y Clases

- Semejanzas
  - Nombre
  - Realizan un conjunto de interfaces
  - Participan en relaciones de
    - Dependencia
    - Generalización y
    - Asociación
  - Anidarse
  - Tienen instancias
  - Participan en interacciones

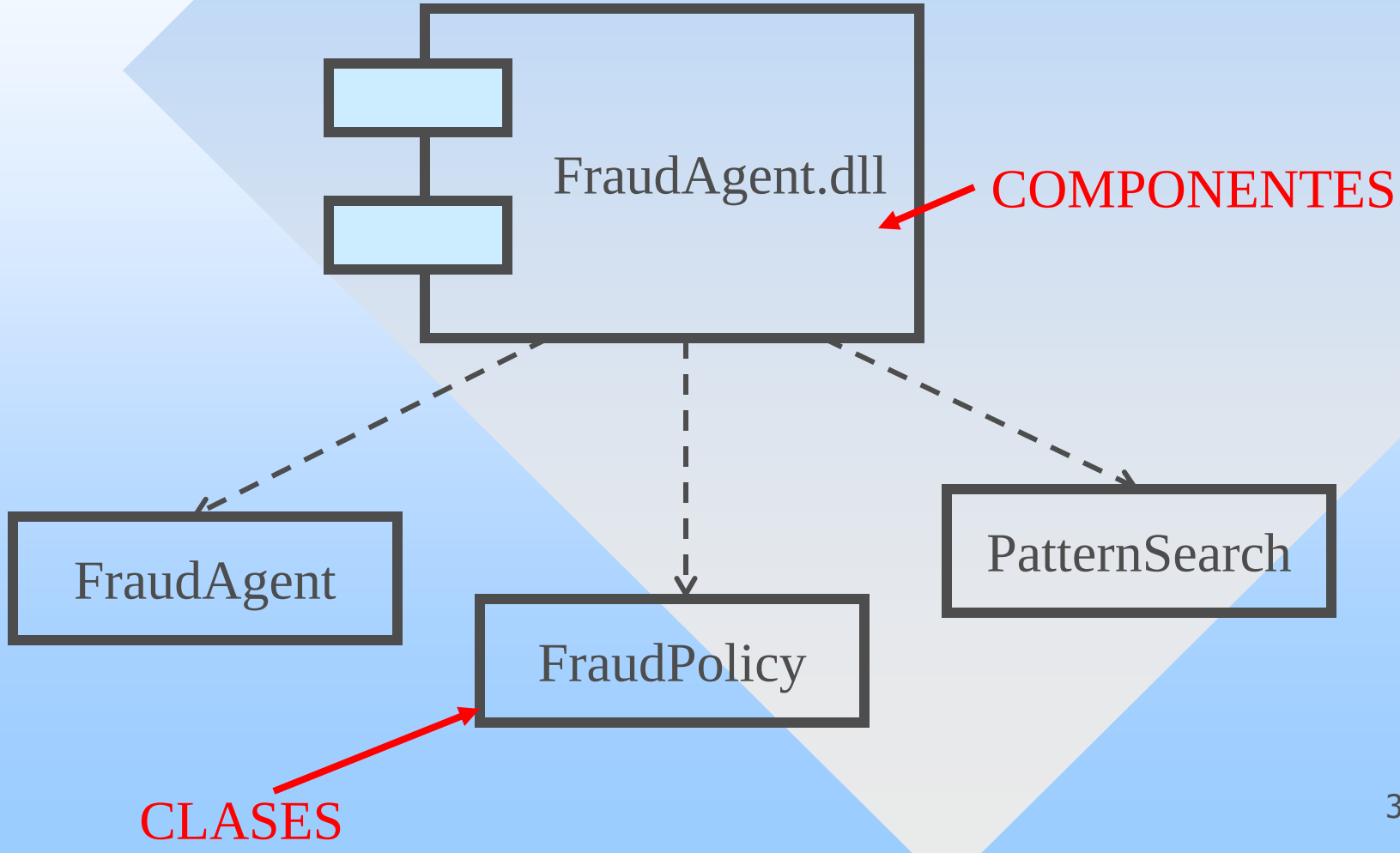
Componentes y Clases

- Representación física
- Empaquetamiento físico
- Operaciones a través de interfaces
- Representación lógica
- Nivel de abstracción
- Acceso a atributos y operaciones

Componentes y Clases
Diferencias

- Residen en un nodo
- Implementación física de un conjunto de elementos lógicos (clases y colaboraciones)
- Servicios disponibles solo por interfaces

- Otro Caso
- Diseño lógico contenible en otros elementos
- Servicios configurables


FraudAgent.dll
COMPONENTES
FraudAgent
FraudPolicy
PatternSearch
CLASES

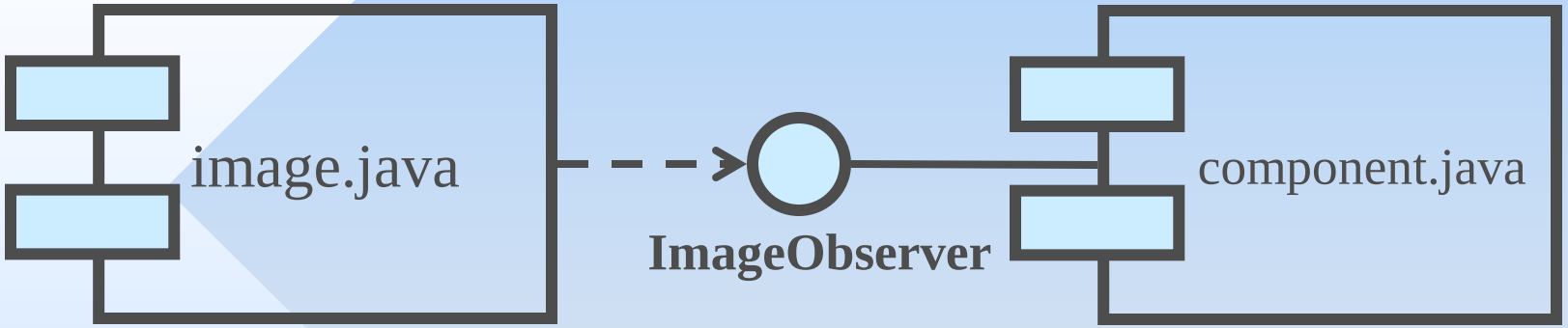
Componentes e Interfaces

- Interfaz
  - Colección de Operaciones que utilizada para especificar un servicio de una clase o componente.
  - Son el pegamento entre los diferentes componentes (se requiere infraestructura basada en componentes del sistema operativo)
    - COM+
    - CORBA
    - Enterprise Java Beans

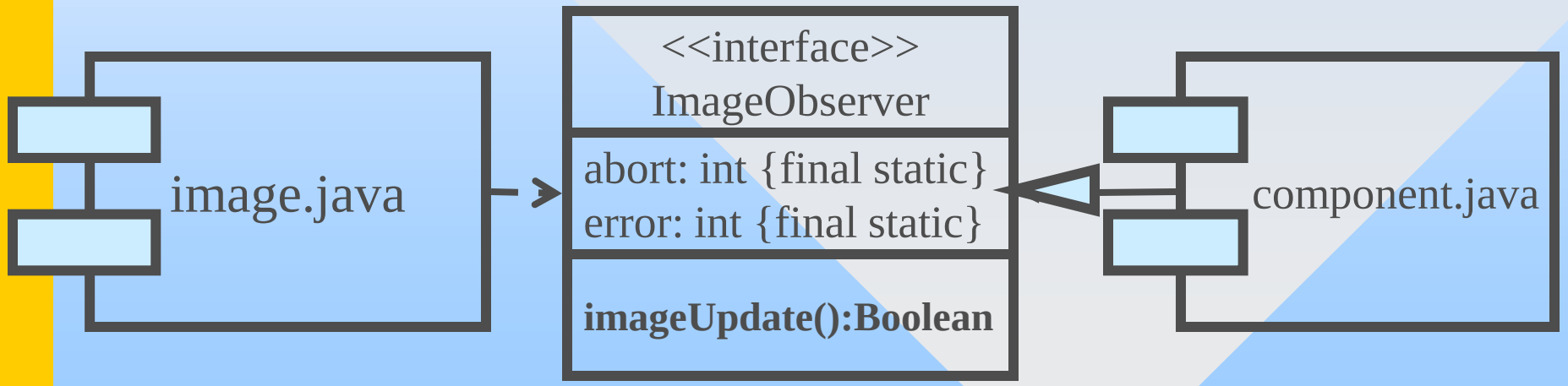
Componentes e Interfaces

- Descomposición de la Implementación Física
- Proveer componentes que realicen las interfaces
- Este mecanismo permite implantar sistemas cuyos servicios son 'location-independent', o remplazables.

FORMA ICÓNICA



FORMA EXPANDIDA



Reemplazo binario

- Partes reemplazables binarias en el Sistema Operativo
- No implican reconstrucción o recompilación inmediata
- Extensibilidad a través de interfaces

Reemplazo binario

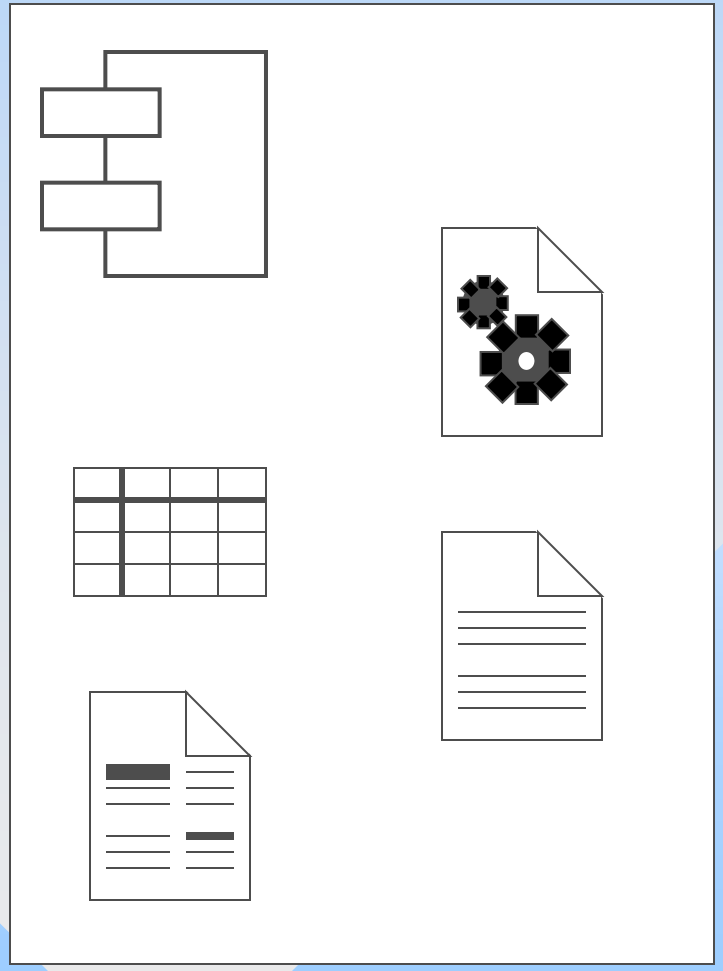
- Un componente es
  - Físico
  - Reemplazable
    - Sustituible
    - Mismas Interfaces
  - Parte del Sistema
    - Arquitectura
    - Tecnología
  - Conforma y provee un conjunto de interfaces

Tipos de Componentes

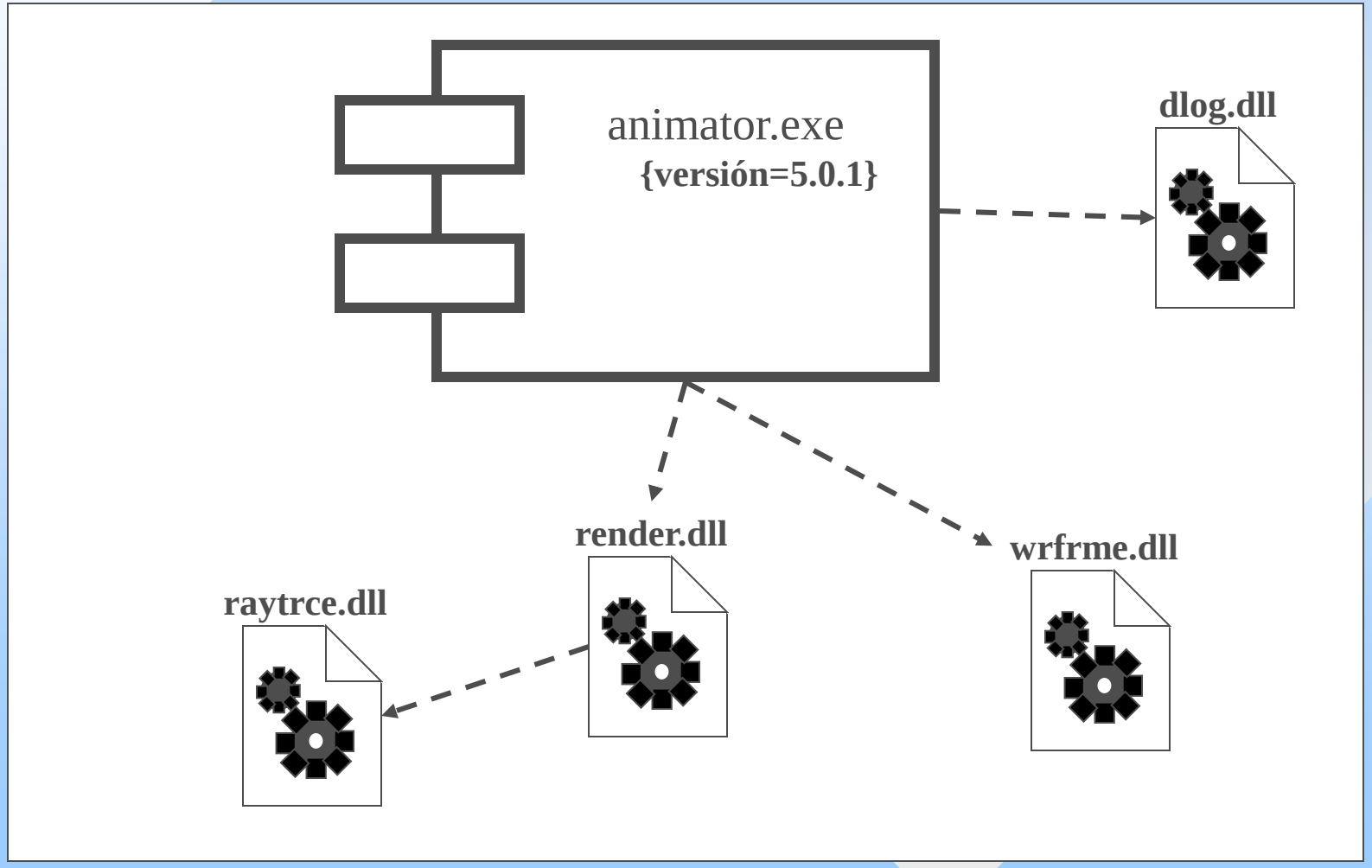
- De Implantación
  - Necesarios y suficientes para generar sistemas ejecutables
    - .EXE
    - .DLL o equivalente
    - Tablas, Páginas Web dinámicas, modelos de objetos, otros ejecutables, etc.
- Productos de Trabajo
  - Fuentes
  - Archivos de Datos
- De Ejecución
  - Aquellos que sólo existen en el ambiente de ejecución
    - Objeto COM+ instanciado

Elementos Estándares

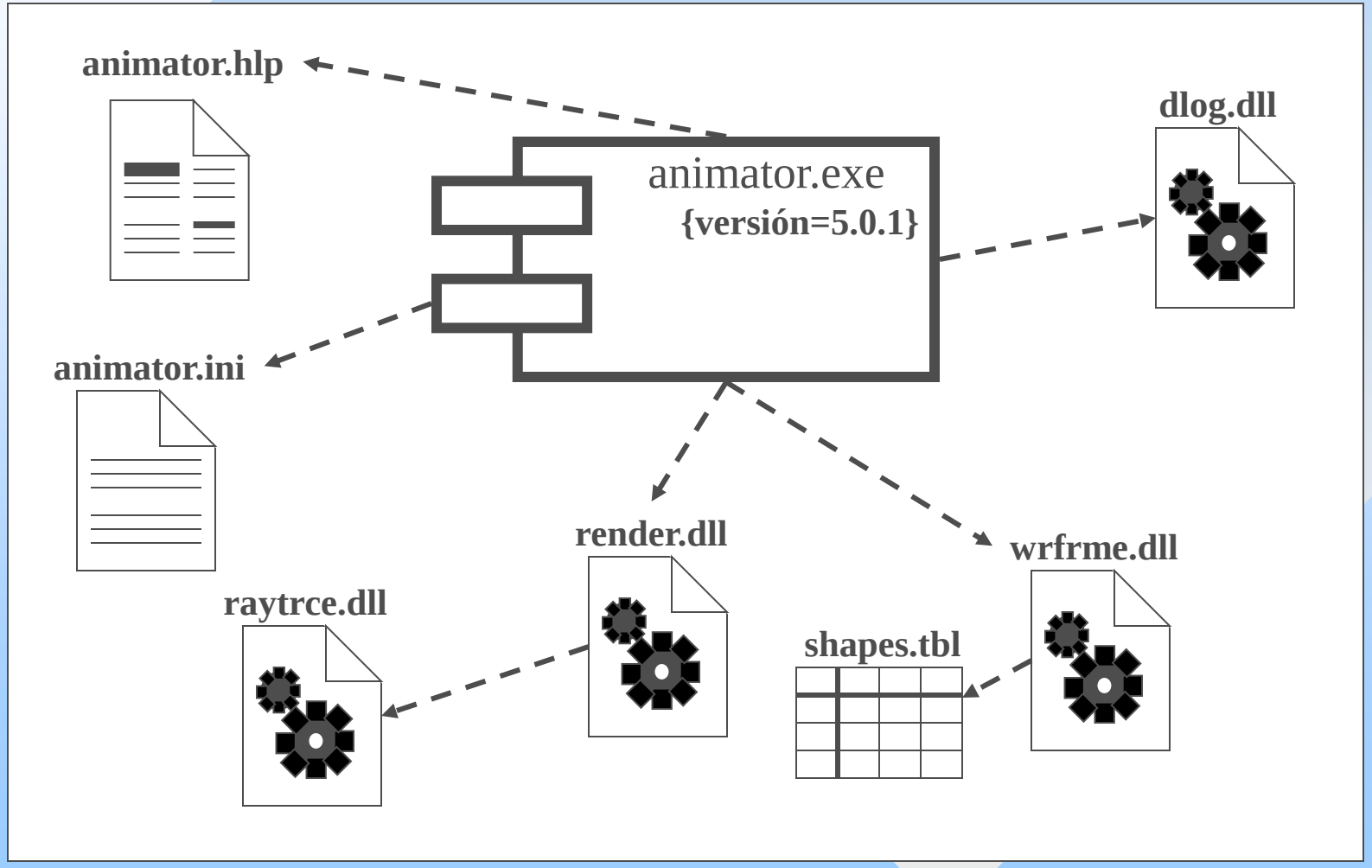
- Ejecutable (*Executable*)
- Biblioteca (*Library*)
  - Estática o Dinámica
- Tabla (*Table*)
- Archivo (*File*)
- Documento (*Document*)



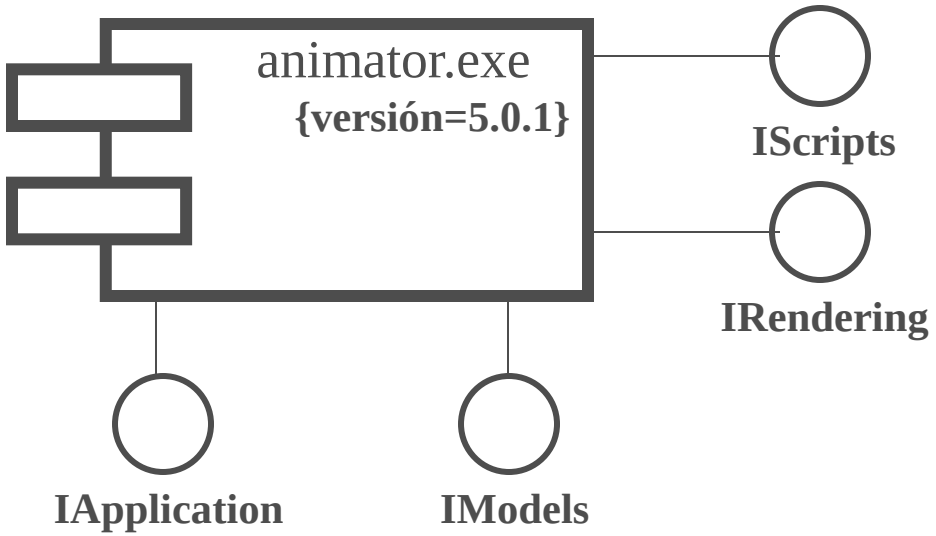
Modelado de Ejecutables y Bibliotecas



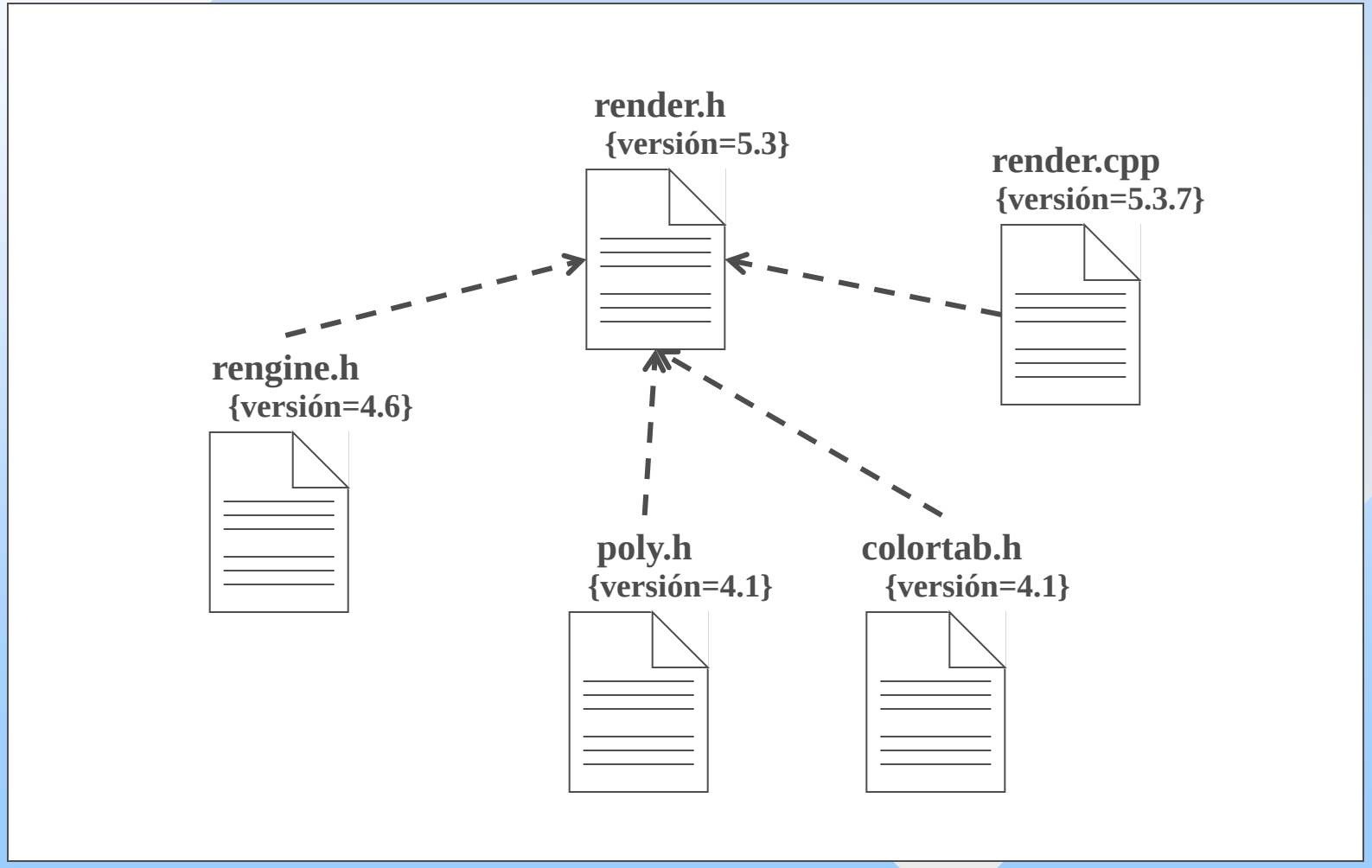
Modelado de Tablas, Archivos y Documentos



Modelado de APIs



Modelado de Código Fuente



EL MODELO DE IMPLANTACIÓN (DIAGRAMAS DE IMPLANTACIÓN o "DEPLOYMENT")

Jorge Kashiwamoto

Panorámica general

- Comprender la utilización de los diagramas de Deployment:
  - Fundamentos
  - Nodos
    - Relación con componentes
    - Paquetes
  - Conexiones
  - Atributos
  - Usos

Diagramas de Deployment

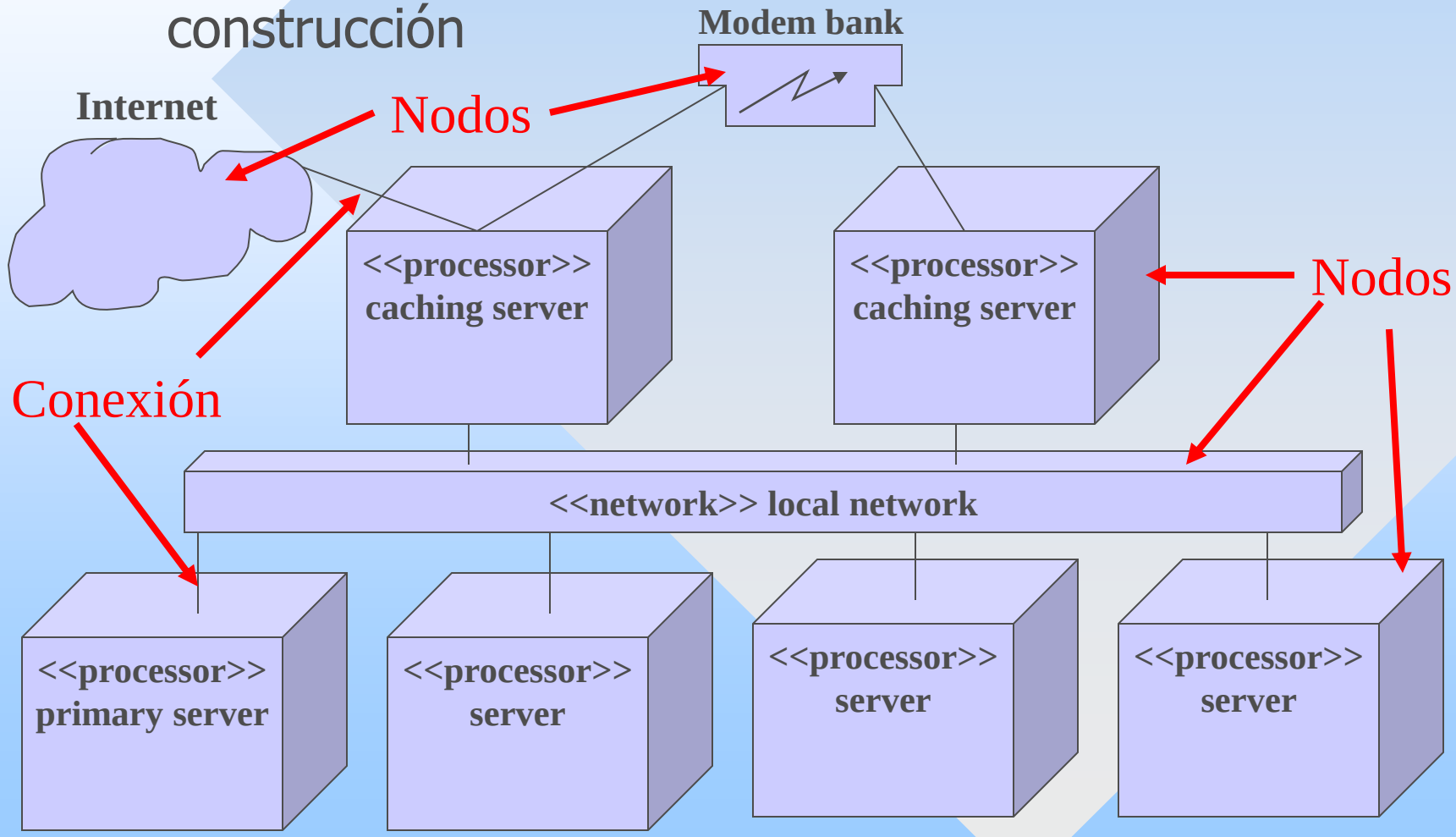
- Modelado de los Aspectos Físicos del Sistema
- Junto con el de componentes son los 2 diagramas en este nivel
- Muestra la configuración de los nodos de procesamiento en tiempo de ejecución y los componentes residentes en ellos

Diagramas de Deployment

- Modela una vista de implantación estática
- Modela la topología del hardware donde se ejecuta la aplicación
- Esencialmente son diagramas de clase enfocados a nodos en el sistema

Diagramas de Deployment

- Visualiza el aspecto estático de los nodos físicos y sus relaciones; así como los detalles de su construcción

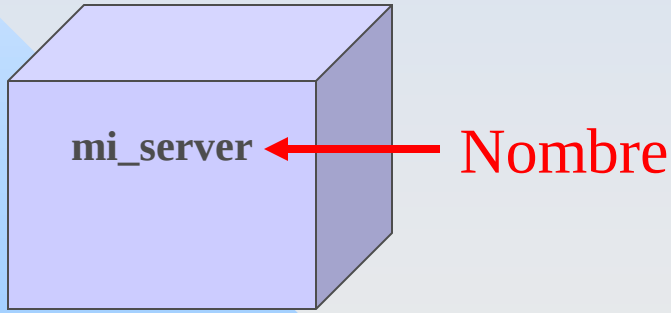


Elementos de un Diagrama de Deployment

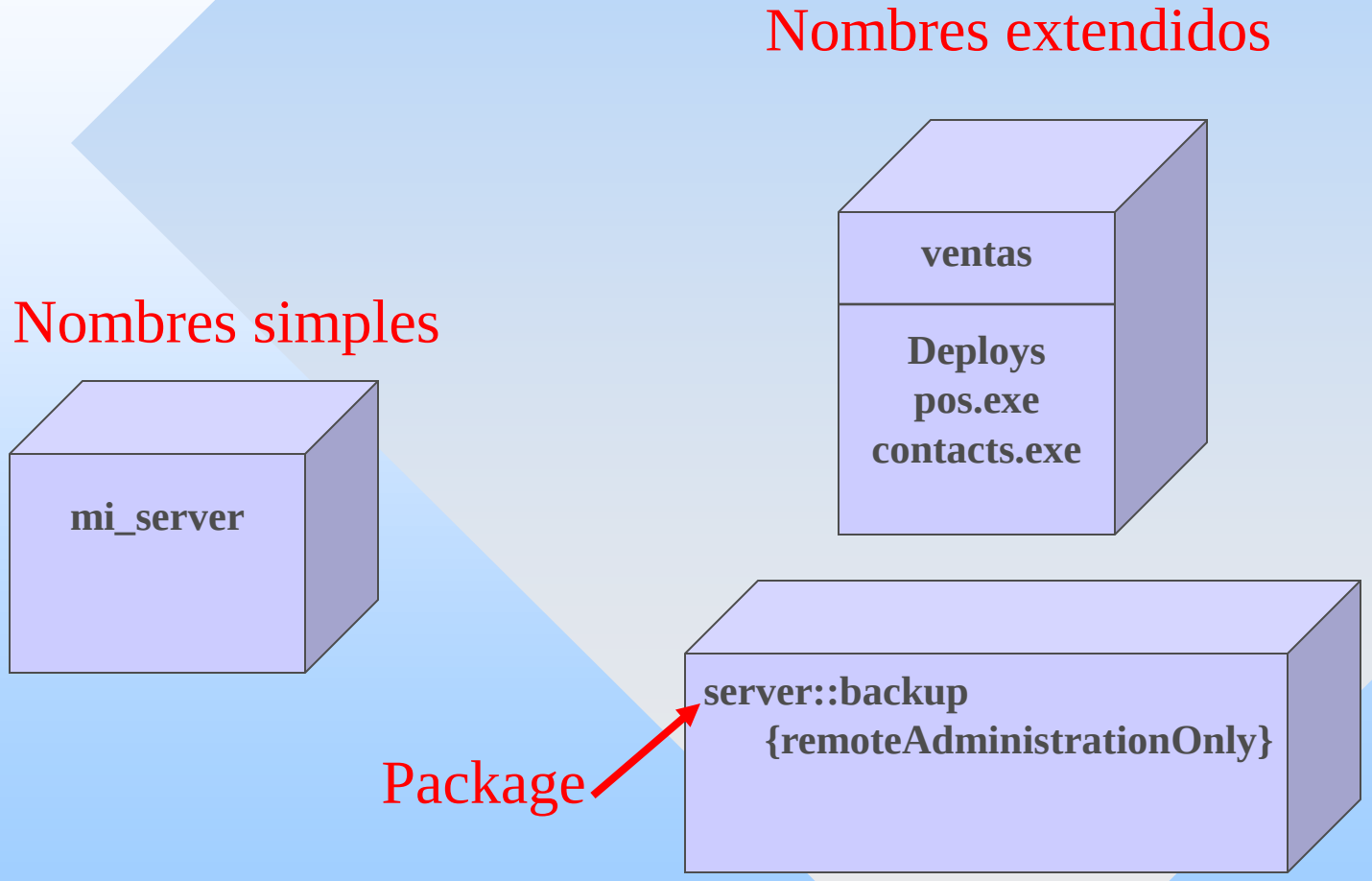
- Nodos
- Relaciones de
  - Dependencia
  - Asociación

Nodo

- Elemento físico que existe en tiempo de ejecución y representa un recurso computacional



Nombres de Nodo



Semejanzas entre Nodos y Componentes

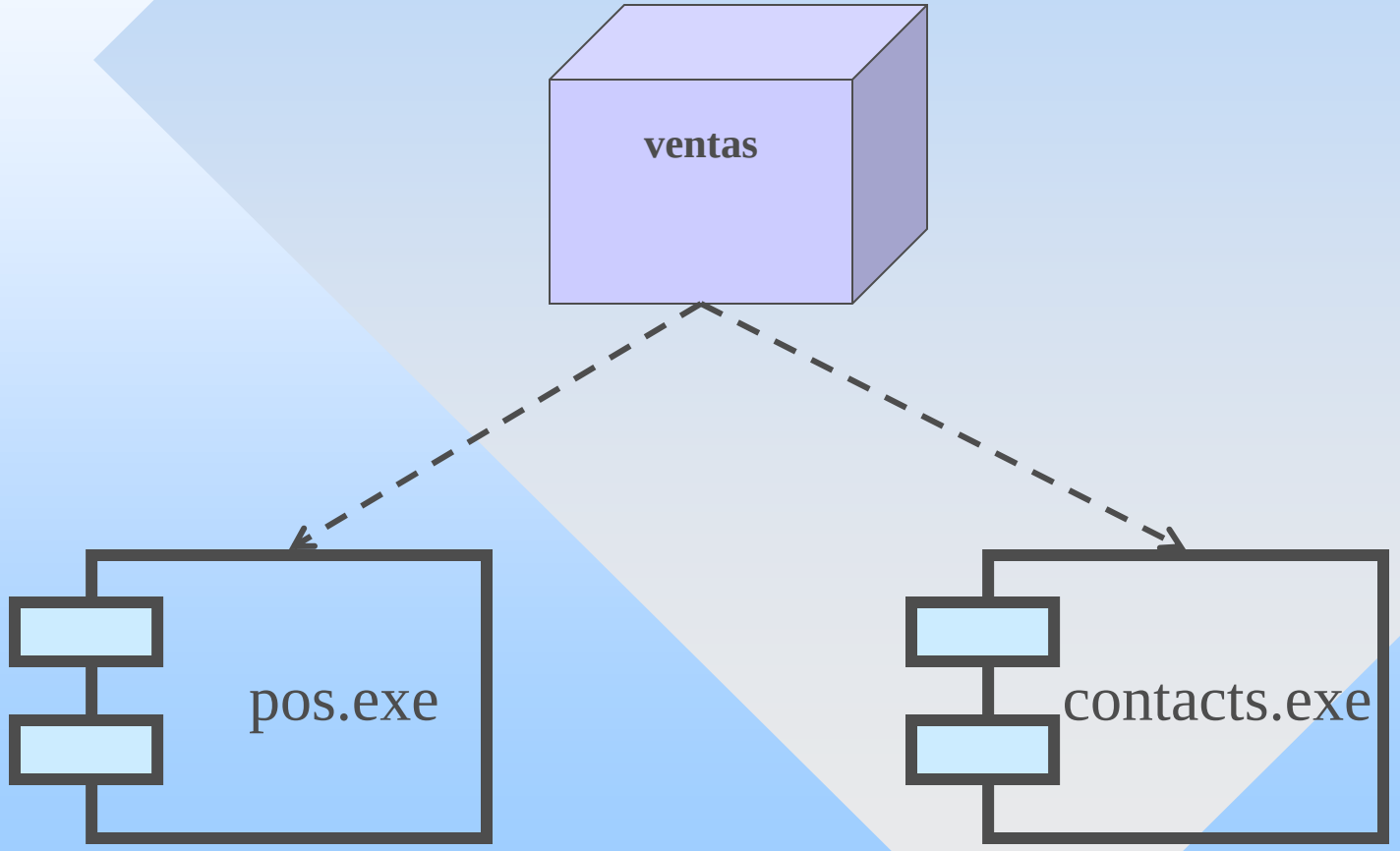
- Nombres
- Relaciones
  - Dependencia
  - Generalización
  - Asociación
- Anidarse
- Tener instancias
- Participar en interacciones

Diferencias entre Nodos y Componentes

- Ejecutan componentes
- Representan la implantación física de componentes

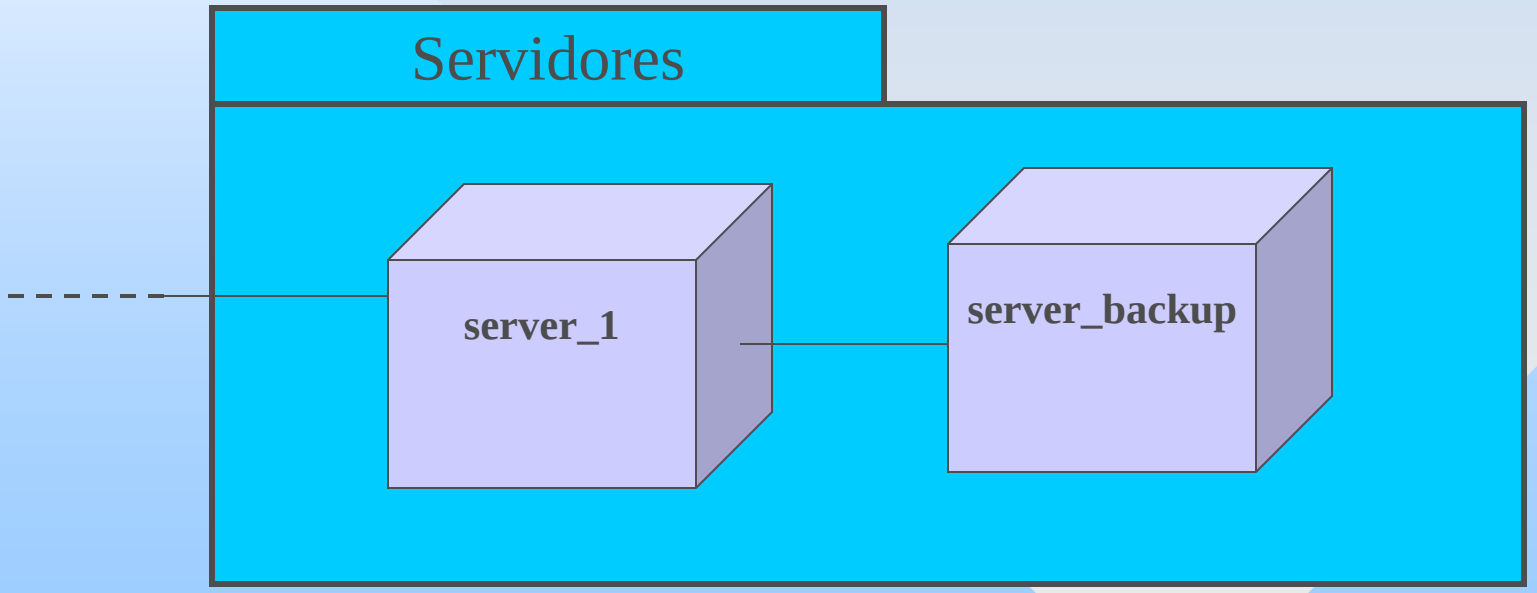
- Participan en la ejecución de un sistema
- Representan el empaquetamiento físico de otros elementos lógicos

Relación entre nodos y componentes



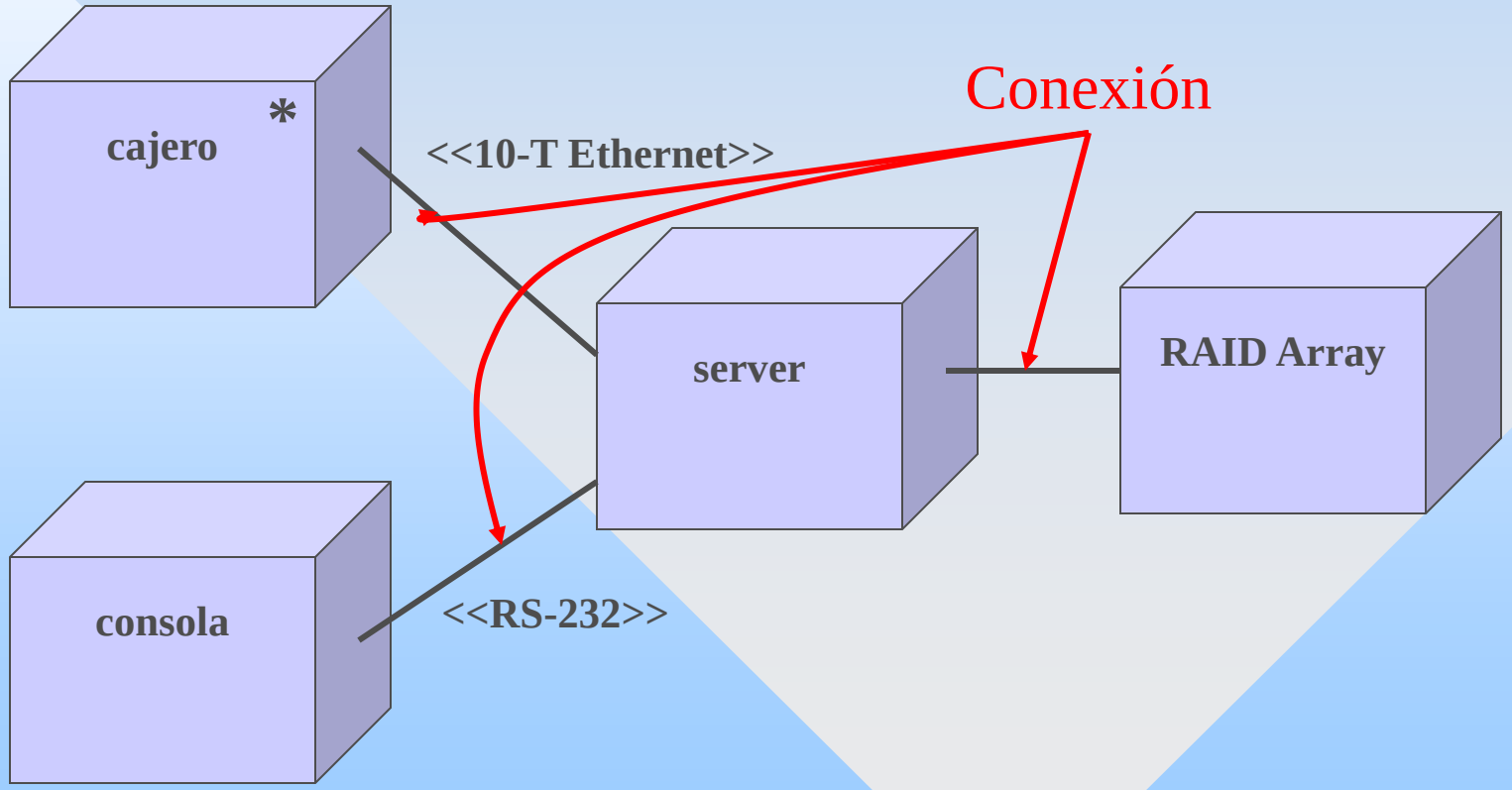
Nodos y Paquetes

- Los nodos pueden ser agrupados en paquetes

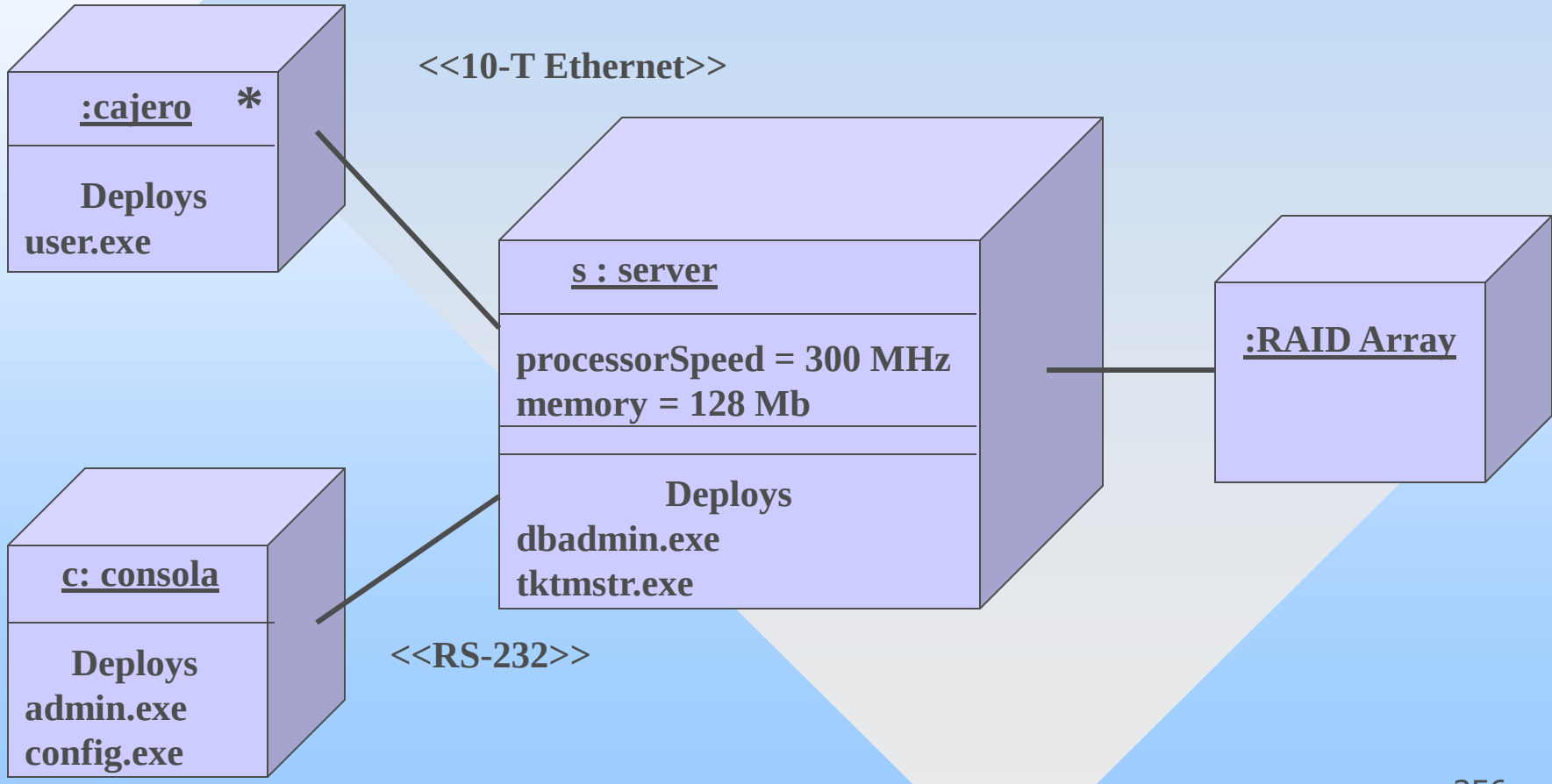


Conexiones

- Asociaciones entre nodos



Ätributos y adornments



Usos comunes

- Modelado de
  - Sistemas Embebidos
  - Sistemas Cliente / Servidor
  - Sistemas Distribuidos

INTRODUCCIÓN AL PROCESO UNIFICADO DE DESARROLLO DE SOFTWARE

Jorge Kashiwamoto

Panorámica general

- Conocer la existencia de una propuesta de proceso unificado de desarrollo
- Conocer la arquitectura del proceso de desarrollo
- Apreciar la importancia, ventajas y desventajas de este tipo de proceso de desarrollo

Metodología

- LENGUAJE
  - V.g.: UML
- PROCESO
  - V.g.: Rational Unified Process
- HERRAMIENTA
  - V.g.: Rose

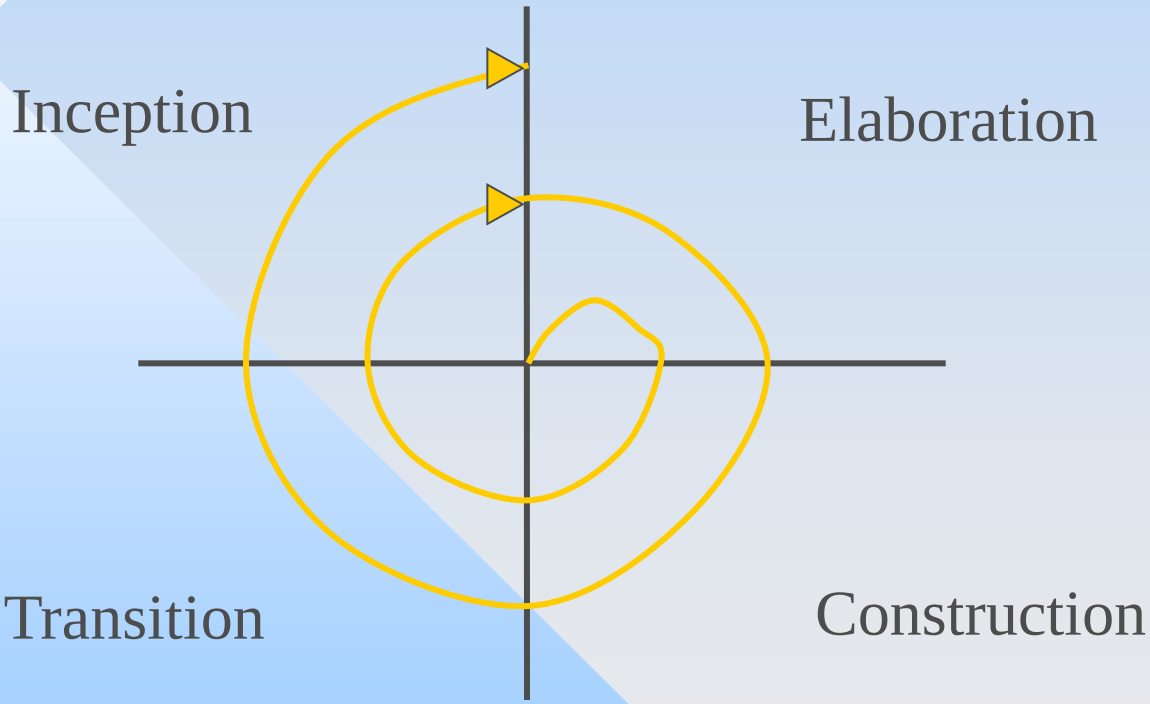
PROCESO

- Elementos de Proyecto
  - Fases
  - Etapas
  - Métodos
  - Técnicas y
  - Prácticas
- Que las personas realizan para desarrollar y mantener software con sus artefactos asociados

RUP

- Rational Unified Process
  - Proceso unificado de desarrollo de software
- Características
  - ***Proceso de Desarrollo de Software***
  - Dirigido por ***Casos de Uso***
  - Centrado en la ***Arquitectura***
  - ***Iterativo*** e ***Incremental***

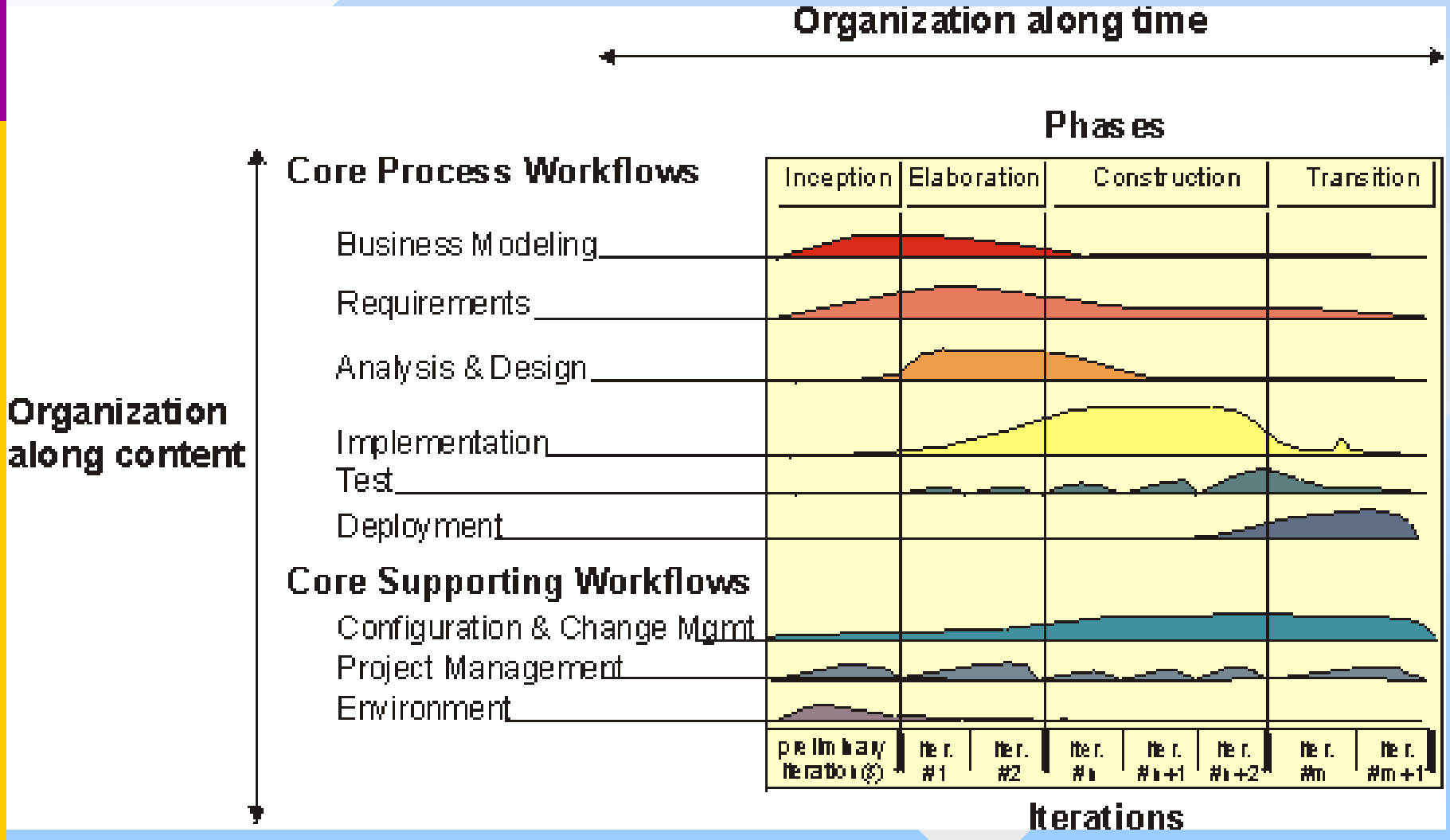
Iterativo e Incremental



Rational Unified Process (RUP)

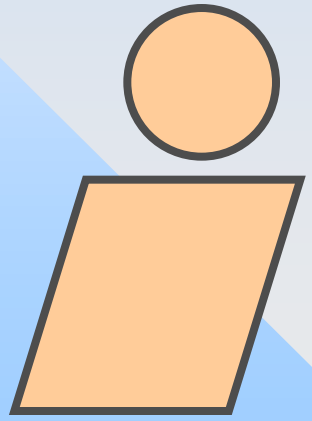
- Objectory Process y otros (Dic98)
  - 4 Fases
  - 9 Flujos de Trabajo
  - Iteraciones

RUP


Organization along time
Phases
Inception
Elaboration
Construction
Transition
Core Process Workflows
Business Modeling
Requirements
Analysis & Design
Implementation
Test
Deployment
Core Supporting Workflows
Configuration & Change Mgmt
Project Management
Environment
Organization along content
preliminary iteration(s)
iter. #1
iter. #2
iter. #n
iter. #n+1
iter. #n+2
iter. #m
iter. #m+1
Iterations

Trabajadores *(Workers)*

- Personas que participan en el proceso
- Tienen competencias específicas
  - En cada flujo de trabajo

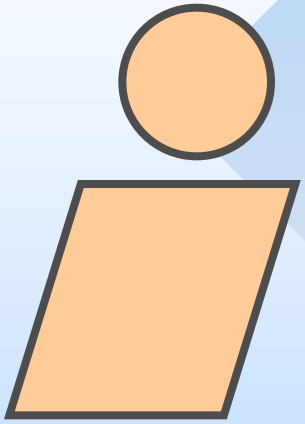


Worker

Trabajadores (2)

- 1. Analista de Sistemas
- 2. Especificador de Casos de Uso
- 3. Arquitecto
- 4. Ingeniero de Casos de Uso
- 5. Ingeniero de Componentes
- 6. Integrador de Sistemas
- 7. Diseñador de Pruebas
- 8. Ingeniero de Pruebas de Integración
- 9. Ingeniero de Pruebas del Sistema

Analista de Sistemas



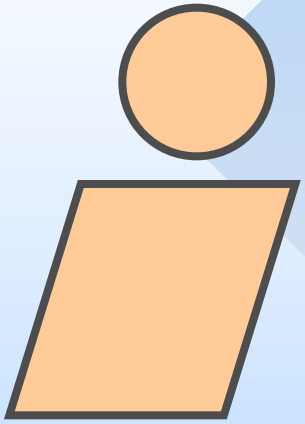
Analista de Sistemas

Flujos de Trabajo

- Requerimientos

- Responsable del conjunto de requisitos modelados en los casos de uso
  - Requisitos funcionales y no funcionales
- Delimitar el sistema
- Encontrar actores y casos de uso
- Asegurar modelos de casos de uso completos y consistentes
- Elaborar un glosario para mantener la consistencia semántica
- Dirigir el modelado
- Coordinar la captura de requisitos

Especificador de Casos de Uso



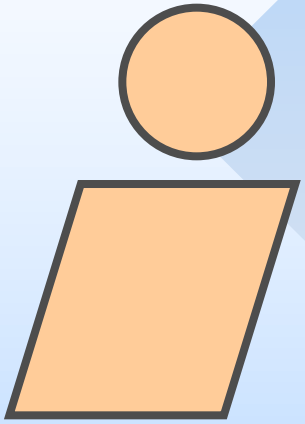
Especificador de C.U.

Flujos de Trabajo

- Requerimientos

- Responsable de la descripción detallada de un caso de uso
- Establecer una comunicación estrecha y eficaz con los usuarios (directos)

Diseñador de Interfaces



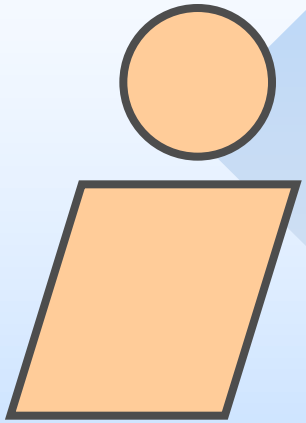
Diseñador de Interfaces

Flujos de Trabajo

- Requerimientos

- Diseñar las interfaces de usuario
  - Aspecto visual
- Desarrollar prototipos de interfaces de usuario para algunos casos de uso

Arquitecto

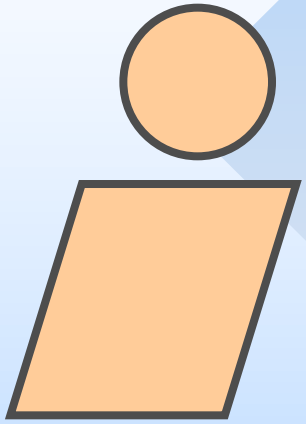


Arquitecto

Flujos de Trabajo
- Requerimientos
- Análisis
- Diseño
- Implantación

- Requerimientos
  - Describir la arquitectura y prioridades del modelo de casos de uso
- Análisis
  - Garantizar la integridad del modelo de análisis
    - Correcto, consistente y legible
- Diseño
  - Garantizar la integridad de los modelos de diseño y despliegue
- Implantación
  - Garantizar la integridad del modelo de implantación
  - Asignar componentes a nodos

Ingeniero de Casos de Uso



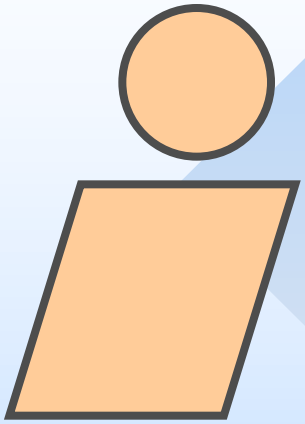
Ingeniero de C.U.

Flujos de Trabajo

- Análisis
- Diseño

- Análisis
  - Mantener la integridad de las realizaciones de casos de uso
- Diseño
  - Detallar las realizaciones de casos de uso
  - Verificar la correspondencia entre análisis y diseño

Ingeniero de Componentes



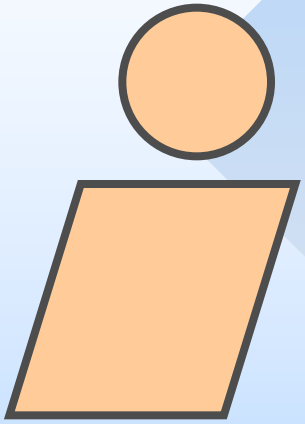
Ingeniero de Componentes

Flujos de Trabajo

- Análisis
- Diseño
- Implantación
- Pruebas

- Análisis
  - Definir y mantener las responsabilidades, atributos, relaciones y requisitos especiales de una o varias clases del análisis
- Diseño
  - Definir y mantener las operaciones, métodos, atributos, relaciones y requisitos de implantación de una o más clases del diseño
- Implantación
  - Definir y mantener el código fuente de uno o varios componentes
- Pruebas
  - Desarrollar componentes de prueba que automatizan algunos de los procedimientos de prueba

Integrador de Sistemas

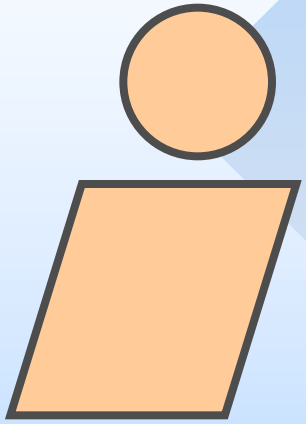


Integrador de Sistemas

Flujos de Trabajo
- Implantación

- Planificar la secuencia de construcciones necesarias en cada iteración
- Integrar cada construcción a partir de sus partes implementadas

Diseñador de Pruebas



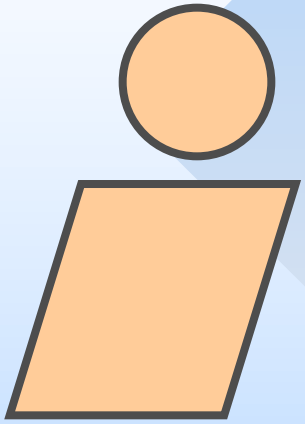
Diseñador de Pruebas

Flujos de Trabajo

- Pruebas

- Garantizar la integridad del modelo de pruebas
- Planear las pruebas
  - Establecer objetivos de prueba apropiados
- Seleccionar y describir casos de prueba y los procedimientos de prueba

Ingeniero de Pruebas de Integración



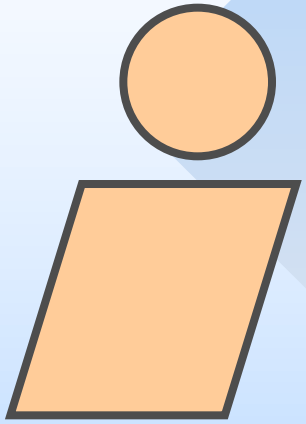
Ing. de Pruebas de Integración

Flujos de Trabajo

- Pruebas

- Ejecutar las pruebas de integración
- Verificar el correcto funcionamiento de componentes
- Documentar los defectos

Ingeniero de Pruebas del Sistema



Ing. de Pruebas del Sistema

Flujos de Trabajo
- Pruebas

- Ejecutar las pruebas del sistema para cada iteración completa
- Verificar los resultados en conjunto con los usuarios finales
- Documentar los defectos
- Facilidad para tener familiaridad con el comportamiento observable del sistema

Ventajas de un Proceso

- Process Know-How → Toma de Decisiones
- Estadarización
- Calidad y Mejores prácticas
- Mantenimiento
- Control de Complejidad
- Control de Proyecto
- Reducción de Costos

Ventajas de RUP

- Aplicación de Principios de Ingeniería
  - Desarrollo
    - Iteratividad
    - Orientado a Requerimientos
    - Basado en Arquitectura
  - Retroalimentación
    - Prototipado
  - Definición de Producto

Desventajas de RUP

- Solamente es un Proceso
- No tiene Soporte Multiproyecto
- Influencia de los Fabricantes de Herramientas
- Carencias en:
  - Métrica
  - Reuso
  - Control de Personal
  - Control de Pruebas