

UD 3

INTRODUCCIÓN AL DISEÑO ORIENTADO A OBJETOS

Pedro J. Ponce de León

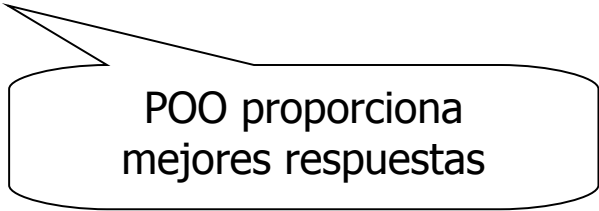
Versión 0.1
(Curso 11/12)



- Diseño O.O.
 - Principios de Parnas
 - Acoplamiento y cohesión
 - Diseño dirigido por responsabilidades (RDD)
 - Principios y artefactos
 - Modelado de objetos
 - Tarjetas CRC
 - Ejercicios de diseño con CRC

- Objetivo principal: conseguir crear un universo de objetos lo más independientes posible entre sí.
 - Una posible técnica a utilizar es la denominada ***Diseño Dirigido por Responsabilidades*** (*responsibility-driven design*) [Wirfs-Brock].

- Pequeños proyectos : 'Programming in the small'
 - Pocos programadores. Un individuo puede abarcar todos los aspectos del proyecto.
 - El mayor problema es el diseño y desarrollo de algoritmos para resolver el problema actual.
- Grandes proyectos: 'Programming in the large' :
 - Gran equipo de programadores. Un individuo no puede hacerse responsable ni es capaz de entender todo el proyecto
 - El mayor problema en el proceso de desarrollo es el manejo de detalles y la comunicación entre las distintas porciones del proyecto



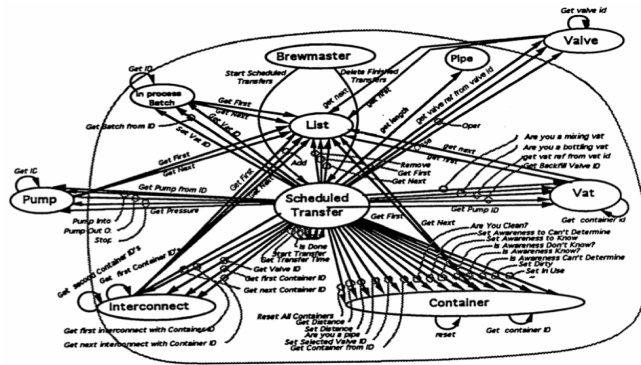
POO proporciona
mejores respuestas

- El énfasis en caracterizar un componente software por su comportamiento tiene una consecuencia fundamental: separación de interfaz (*qué*) e implementación (*cómo*).
- ***Principios de Parnas***
 - El desarrollador de un componente sw C debe proporcionar al usuario de C toda la info necesaria para hacer un uso efectivo de los servicios de C y no debería proporcionar ninguna otra información.
 - El desarrollador de un componente sw C debe recibir toda la información necesaria para realizar las responsabilidades necesarias asignadas al componente y ninguna otra información.

- **Acoplamiento:** relación entre componentes software
 - Interesa bajo acoplamiento. Éste se consigue moviendo las tareas a quién ya tiene habilidad para realizarlas.
-
- **Cohesión:** grado en que las responsabilidades de un solo componente forman una unidad significativa.
 - Interesa alta cohesión. Ésta se consigue asociando a un solo componente tareas que están relacionadas en cierta manera, p. ej., acceso a los mismos datos.

- Uno de los principios básicos de la ingeniería del software es **“incrementar la cohesión, reducir el acoplamiento”**.
- Componentes con bajo acoplamiento y alta cohesión facilitan su utilización y su interconexión.

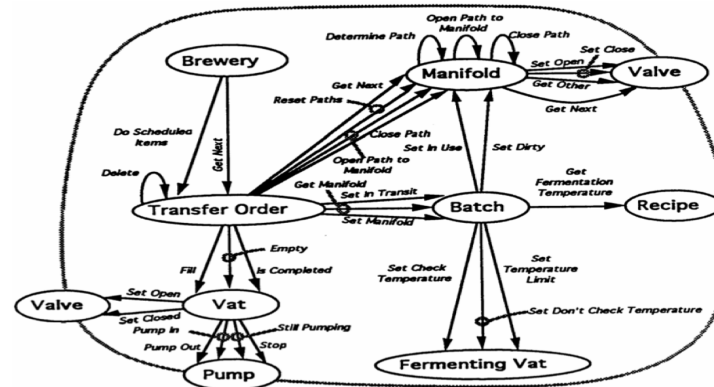
Acoplamiento vs. cohesión



sistema demasiado acoplado

Acoplamiento

sistema desacoplado y más cohesionado

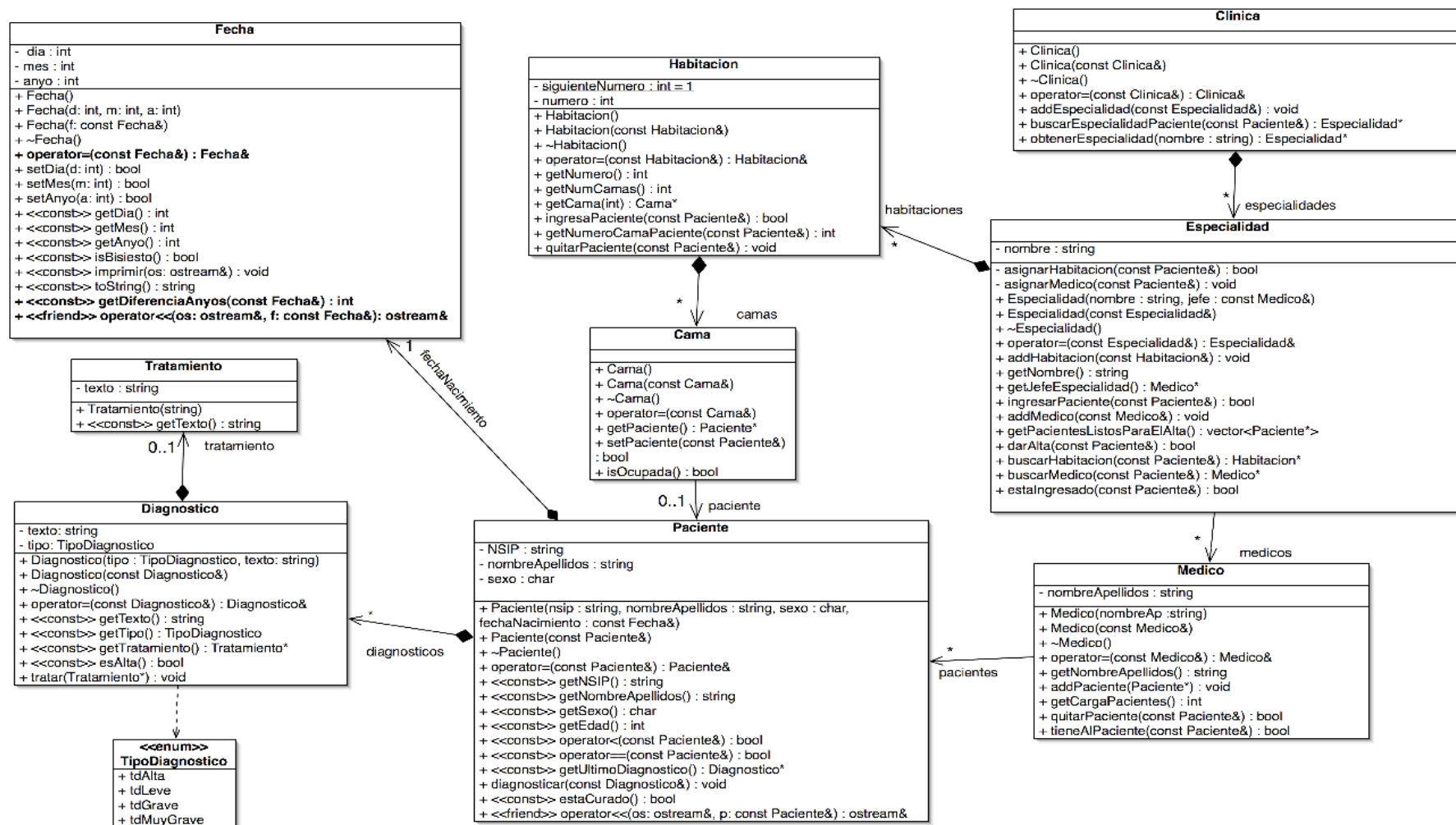


Cohesión

- Un diagrama de clases define las clases, sus propiedades y cómo se relacionan unas con otras.
- Proporciona una vista **estática** de los elementos que conforman el software.
 - Se ven las partes que componen la aplicación y cómo se ensamblan, pero no cómo se comportan cuando el sistema se ejecuta.
- El diagrama de clases es el diagrama principal de análisis y diseño.

El diseño de Aplicaciones OO

Diagrama de Clases (UML)



Responsibility-Driven Design (RDD)

- Proporciona técnicas informales para el modelado de roles, responsabilidades y colaboraciones de objetos.

Ejemplo: diseñar un caballo

- Objetos: Cabeza, cuerpo, cola, patas (4)
- Comportamiento: Arrancar, parar, acelerar, decelerar, ...

Diseñar un caballo con responsabilidad:

- ¿Para qué queremos al caballo?
 - Para pasear
 - Para competir
 - Para rejonear

El diseño de Aplicaciones OO

- Principios de RDD



- **Maximizar la abstracción**

Pensar en las responsabilidades de los objetos de ‘conocer/saber’, ‘hacer’ y ‘decidir’: ¿quién es responsable de saber tal o cual cosa? ¿Quién es responsable de hacer esto o lo otro? Para esto necesitamos a un experto en el dominio (podemos ser nosotros mismos)

- **Distribuir el comportamiento**

Promueve una arquitectura de control delegado (que una tarea determinada la haga quien sabe hacerlo. Por ejemplo, ¿quién es responsable de imprimir una fecha en formato DD/MM/AAAA?

- **Crear objetos ‘inteligentes’,**
que sepan hacer ciertas cosas, que no sean sólo un capazo donde guardar datos.
- **Preservar la flexibilidad**
Flexibilidad: capacidad de algo de ser modificado y adaptado a los cambios del entorno.

Diseña objetos de forma que los detalles internos se puedan modificar fácilmente, sin afectar a otros objetos o al sistema.

- **Aplicación** = un conjunto de objetos interactivos
- **Objeto** = una implementación de uno o más roles
- **Rol** = un conjunto de responsabilidades
- **Responsabilidad** = la obligación de realizar una tarea o conocer cierta información
- **Colaboración** = una interacción entre objetos o roles (o ambos)

- **Modelado de objetos:**

1. Identificar las clases de objetos que forman parte del sistema
2. Asignar responsabilidades a cada clase de objeto

- **Consejo:**

Describe el sistema a modelar como si contaras una historia/escribieras un artículo

- Identifica los temas importantes
- Encuentra candidatos/protagonistas
- Identifica los **sustantivos** y **verbos** de la descripción del sistema que pueden corresponderse con objetos y su comportamiento.

■ Pautas para modelar objetos

- Busca los conceptos clave del dominio: conceptos familiares para los expertos en el dominio/problema a resolver
- Elige nombres con sentido
- Distingue objetos con diferentes comportamientos.
- Coloca a los objetos en su contexto

Asignar **NOMBRES** a los objetos

¡Esto es importante!

- El nombre debería encajar con el rol del objeto en el sistema.
Por ej. un proveedor de servicio es algo que realiza un trabajo: StringTokenizer, ClassLoader, Authenticator.
- El nombre debe dar al programador que utilizará esos objetos una idea suficientemente clara de lo que el objeto sabe hacer (sin entrar en detalles):
TemporizadorConUnaPrecisionDeMasMenosDosMilisegundos → (mejor simplemente Temporizador)
- Capitalización o subrayado para separar palabras:
LectorDeTarjetas o Lector_de_tarjetas
 - Ojo con las abreviaturas, no todo el mundo las entiende y pueden confundir:
TermProcess

Asignar **NOMBRES** a los objetos

- Evita nombres con varias interpretaciones posibles:
 vacío() : ¿nos dice si el objeto está vacío o lo vacía?
- No uses dígitos
- Booleanos: usa nombres que permitan interpretar claramente su valor:
ImpresoraPreparada es mejor que EstadoImpresora.
- Un buen lugar para ver ejemplos de nombres informativos es el API de Java.
Úsalos como modelo.

El diseño de Aplicaciones OO

RDD: Modelado de objetos (CRC)



Ejercicio: Intenta averiguar (sin consultar el API de Java) qué hacen estos tipos de objetos :

ProcessBuilder
SecurityManager
StringBuffer
ArithmeticException
NullPointerException
NoClassDefFoundError
Comparator
EventListener
SortedMap
PropertyPermission
ImageReader
ImageWriter
ContainerOrderFocusTraversalPolicy (AWT)
MenuShortcut

Selecciona candidatos que puedes

- Nombrar
- Definir su propósito
- Asignarle una o dos responsabilidades
- Comprender cómo los demás lo ven

Una **responsabilidad** es algo que una clase conoce o hace.

P. ej.: Un Estudiante conoce su nombre, su DNI, su domicilio. Un Estudiante se matricula en asignaturas, se presenta a exámenes, solicita certificados,...

Las responsabilidades son algo más que una sola operación o atributo: a menudo involucran varias operaciones y/o manejan varios atributos o ítems.

Usa descripciones claras y ‘contundentes’: Usa verbos con un significado claro: ‘eliminar’, ‘mezclar’, ‘calcular’, ‘activar’, en lugar de otros como ‘organizar’, ‘mantener’, ‘procesar’, ‘aceptar’...

Consejos para asignar responsabilidades

- Mantener el comportamiento junto a la información que maneja en el mismo objeto
- No crear objetos (roles) demasiado grandes. Esto los hace más comprensibles
- Distribuye la inteligencia del sistema
- Una misma responsabilidad puede ser implementada por uno o más métodos
- DELEGA: si es posible, haz que otros objetos hagan parte del trabajo.

Colaboración:

Como un grupo de objetos trabaja conjuntamente para realizar una tarea.

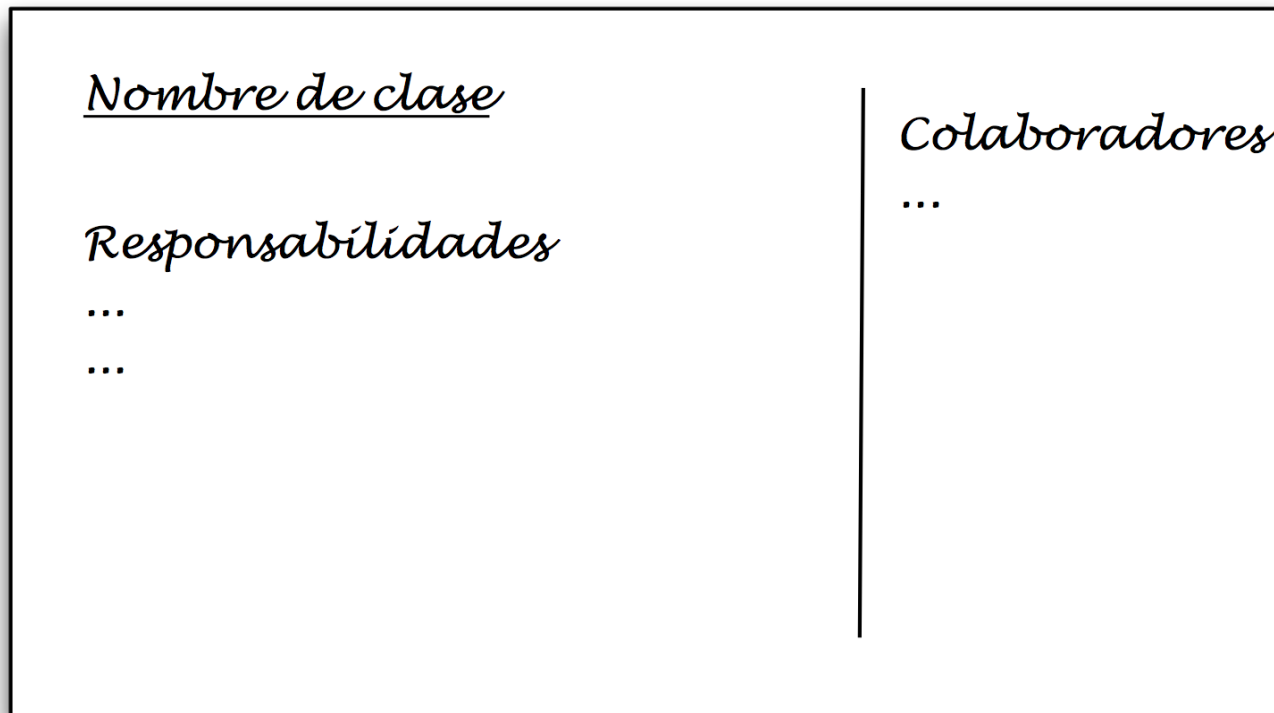
La lista de colaboradores de una clase debe incluir todas aquellas clases que necesita conocer.

- Obligatoriamente, las que suministran servicios que la clase actual necesita para llevar a cabo alguna de sus responsabilidades
- Opcionalmente, aquellas que utilizan servicios proporcionados por la clase actual.

El diseño de Aplicaciones OO

Tarjetas CRC

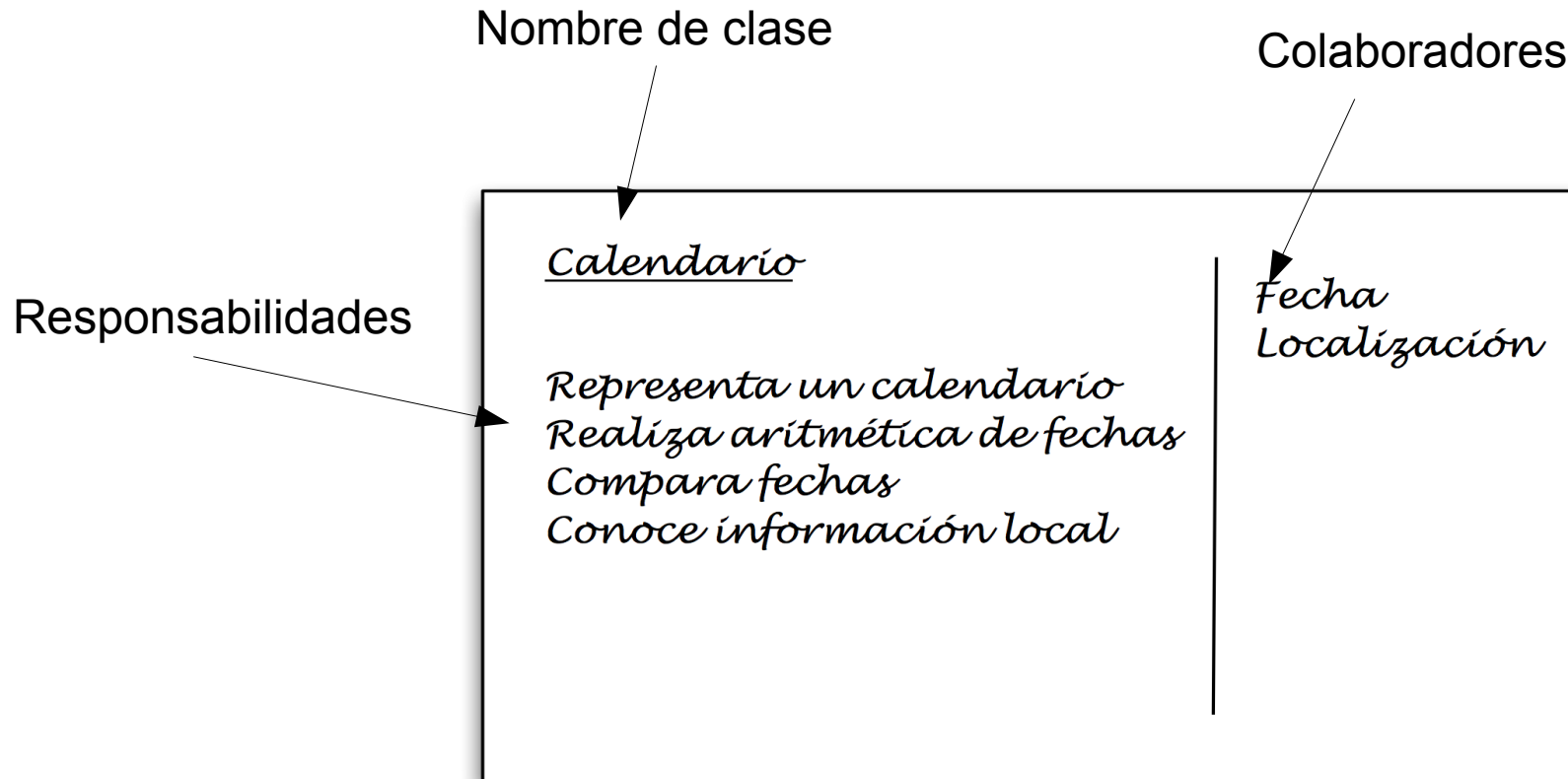
CRC: Class, Responsibility, Collaborators (Clase, Responsabilidades, Colaboradores)



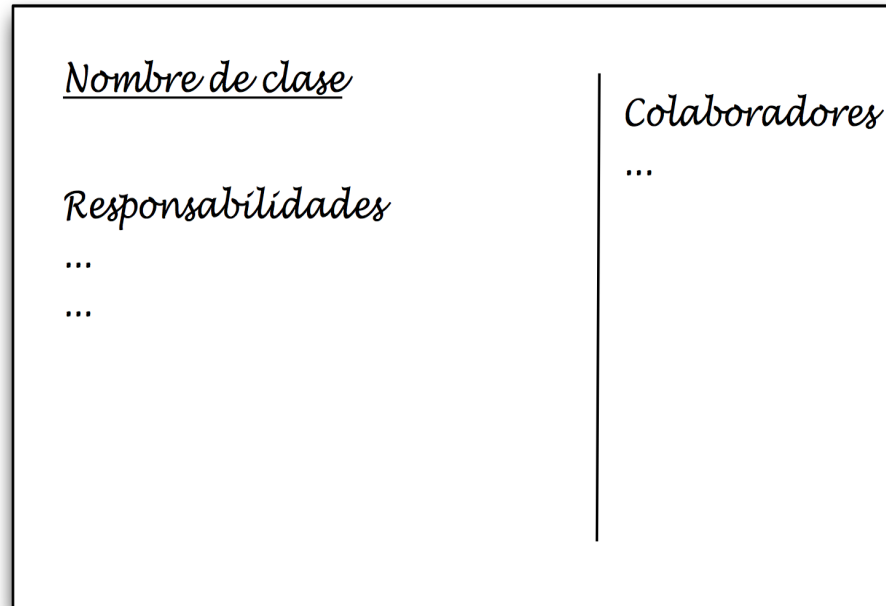
Toda la información sobre una clase de objetos es escrita sobre una tarjeta índice, de las usadas (antiguamente) en las bibliotecas.

El diseño de Aplicaciones OO

Tarjetas CRC



Tarjeta CRC



Nombre de la clase: crea un vocabulario para discutir un diseño. Debemos encontrar un conjunto nombres consistente y evocativo que describa a nuestros objetos en el contexto de nuestra aplicación.

Las **responsabilidades** identifican tareas que se deben resolver. Se expresan mediante un puñado de frases simples, cada una con un verbo activo.

Colaboradores: "no object is an island" Todos los objetos establecen relaciones con otros. Llamamos objetos colaboradores a aquellos que recibirán mensajes o enviarán mensajes al objeto actual, con objeto de satisfacer una responsabilidad.

Modelado de objetos con tarjetas CRC

Las tarjetas CRC enfatizan la cohesión y el (des)acoplamiento. El tamaño de la tarjeta es una buena aproximación a la complejidad que debería tener un objeto.

Las tarjetas se pueden

- Apilar: partes debajo del todo

- Disponer unas cerca de otras (colaboradores cercanos)

- Disponerlas por capas

- ...

En definitiva, hacer lo que consideres conveniente para ilustrar con ellas cómo se comportan los objetos

Modelado de objetos con tarjetas CRC

Es muy útil plantearse escenarios de ejecución, donde imaginamos al sistema a diseñar en ejecución

- 1. ¿Qué es lo que hay que hacer?**
- 2. ¿Quién tiene que hacerlo?**

El diseño de Aplicaciones OO

Ejercicio de diseño con tarjetas CRC



Sistema de venta por catálogo.

Queremos crear una aplicación para un sistema de venta por catálogo. El catálogo contiene una lista de productos con su nombre, una descripción, su precio y una referencia de catálogo. Nos interesa que el sistema lleve un control del stock actual, para saber si un producto determinado se puede vender en el momento que se pide o no. Los clientes acuden a nuestra tienda, consultan el catálogo y nos indican qué productos del catálogo quieren comprar y en qué cantidad. Una vez comprobado que hay stock suficiente de cada producto, emitimos un ticket con el detalle de la compra y su importe total, el cual entregamos al cliente junto a la lista de productos solicitados.

El diseño de Aplicaciones OO

Ejercicio de diseño con tarjetas CRC



Cajero automático (dispensador de billetes)

Queremos crear una aplicación que gestione un cajero automático en el que únicamente se puede sacar dinero mediante tarjeta de débito. Cuando un cliente usa el cajero, lo primero que hace es introducir su tarjeta bancaria. El sistema le pide la contraseña y, si es correcta, le solicita a continuación la cantidad de dinero que desea retirar (sólo se permiten múltiplos de 5). Cada tarjeta tiene asociado un límite diario para sacar dinero con ella. Si la cantidad solicitada es inferior a dicho límite, el cajero comprueba que el saldo de la cuenta bancaria asociada a la tarjeta es suficiente para deducir de ella el dinero a retirar. En caso afirmativo, se deduce la cantidad solicitada del saldo de la cuenta y se le entrega al cliente la cantidad en billetes de 50,20,10 y/o 5 euros, a través del dispensador de billetes, y un justificante de la operación, que muestra, además de la cantidad retirada, el saldo restante en la cuenta y en el límite diario de la tarjeta. Una vez el cliente ha retirado el dinero y el justificante, se le devuelve la tarjeta. Si el cliente tarda más de un minuto en retirar el dinero, este se vuelve a introducir en el cajero y se anula la operación. Si, una vez retirado el dinero, el cliente tarda más de un minuto en retirar la tarjeta, esta se vuelve a introducir en el cajero, quedando custodiada en él para su posterior reclamación por parte del cliente.

-
- T. Budd. ***An Introduction to Object-oriented Programming, 3rd ed.***
 - Cap. 3
- Rebecca Wirfs-Brock, **A Brief Tour of Responsibility-Driven Design**
 - http://wirfs-brock.com/PDFs/A_Brief-Tour-of-RDD.pdf
- Kent Beck, **A Laboratory for Teaching Object-Oriented Thinking**
 - <http://c2.com/doc/oopsla89/paper.html>