PROG-3

SUBTOPIC 3
INTRODUCTION TO OBJECT-ORIENTED DESIGN

Pedro J. Ponce de León

Translation into English by Juan Antonio Pérez
(in most cases just borrowing other's words in the bibliography)

Version 0.1
(Academic Year 2011-2012)

lsi Departamento de Lenguajes y Sistemas Informáticos

Universitat d'Alacant
Universidad de Alicante

Contents

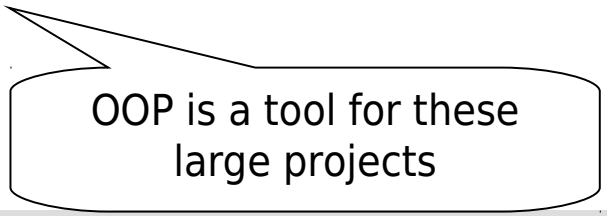
- OO design:
  - Parna's principles
  - Coupling and cohesion
  - Responsibility-driven design
  - CRC cards
  - Exercises

OO Application Design

- The most important aspect of OOP is the creation of a universe of largely autonomous interacting agents.
  - How does one come up with such a system?
  - One possible technique is is termed **responsibility-driven design** [Wirfs-Brock].

OO Application Design

- Individual projects : ‘Programming in the small’
  - Code is developed by a single programmer, or perhaps by a very small collection of programmers. A single individual can understand all aspects of a project.
  - The major problem in the software development process is the design and development of algorithms for dealing with the problem at hand.
- Large projects: ‘Programming in the large’
  - The software system is developed by a large team, often consisting of people with many different skills. No single individual can be considered responsible for the entire project, or even necessarily understands all aspects of the project.
  - The major problem in the software development process is the management of details and the communication of information between diverse portions of the project.



OO Application Design
Interface and implementation

- The separation of interface (what) and implementation (how) is perhaps the most important concept in software engineering.

- ***Parna's principles***
  - The developer of a software component must provide the intended user with all the information needed to make effective use of the services provided by the component, and should provide no other information.

  - The developer of a software component must be provided with all the information necessary to carry out the given responsibilities assigned to the component, and should be provided with no other information.

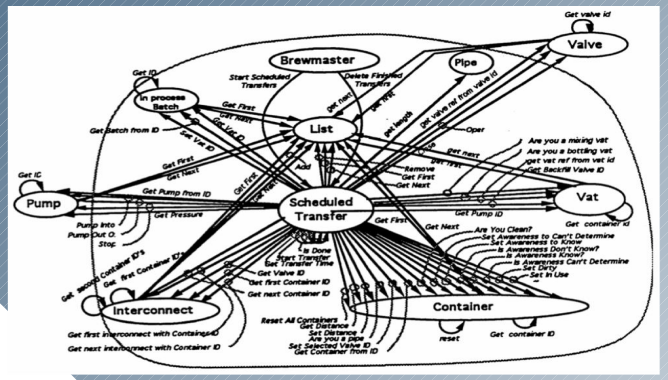
OO Application Design
Quality metrics

- **Coupling** describes the relationship between software components.
  - Low coupling (desirable) may be attained by moving a task into the list of responsibilities of the component that holds the necessary data.
- **Cohesion** is the degree to which the responsibilities of a single component form a meaningful unit.
  - High cohesion (desirable) is achieved by associating in a single component tasks that are related in some manner; e.g. the necessity to access a common data value.

OO Application Design
Quality metrics

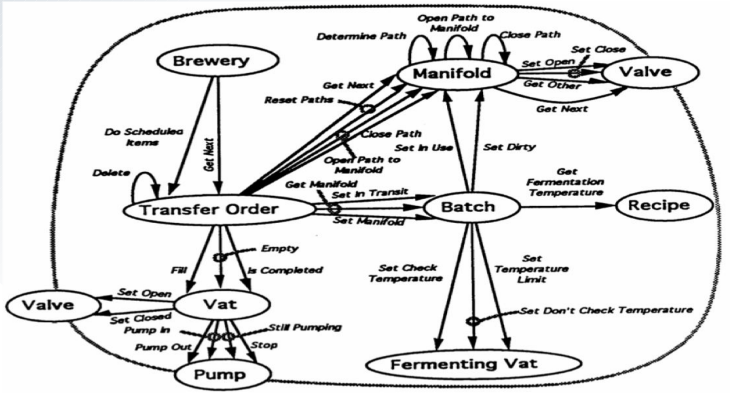
- **Low coupling** is often a sign of a well-structured computer system and a good design, and when combined with **high cohesion**, supports the general goals of high readability and maintainability.

Coupling vs. Cohesion



sistema demasiado acoplado

sistema desacoplado y más cohesionado



Acoplamiento

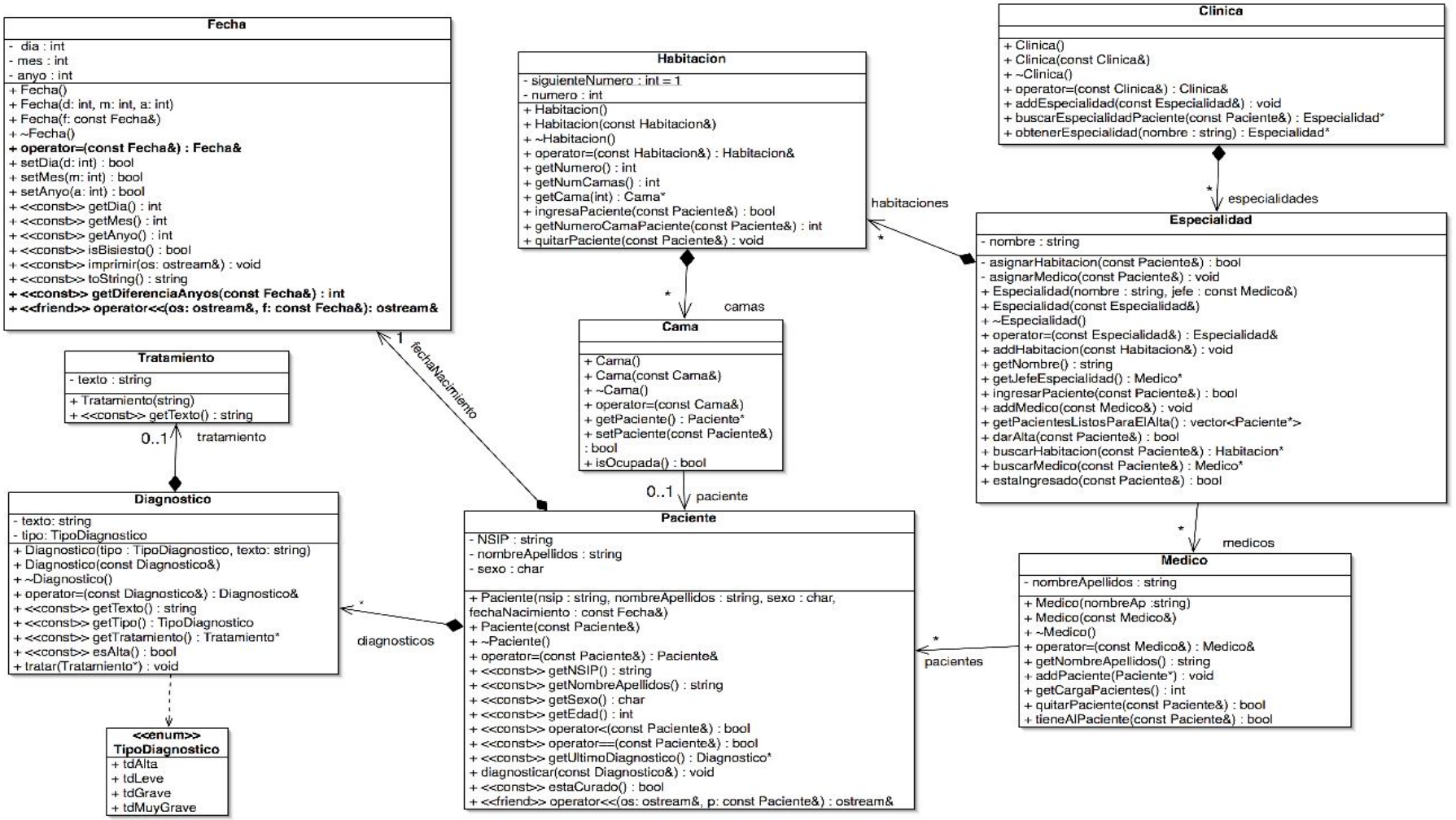
Cohesión

OO Application Design
Class Diagram (UML)

- A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes.
  - Components of an application and their interactions are shown, but inner details are hidden.

- The class diagram is the main building block of object oriented modeling. It is used both for general conceptual modeling of the systematics of the application, and for detailed modeling translating the models into programming code.

OO Application Design
Class Diagram (UML)



OO Application Design
Responsibility-Driven Design

Responsibility-Driven Design (RDD)

- An informal way to design software that emphasizes modeling of objects' roles, responsibilities, and collaborations.

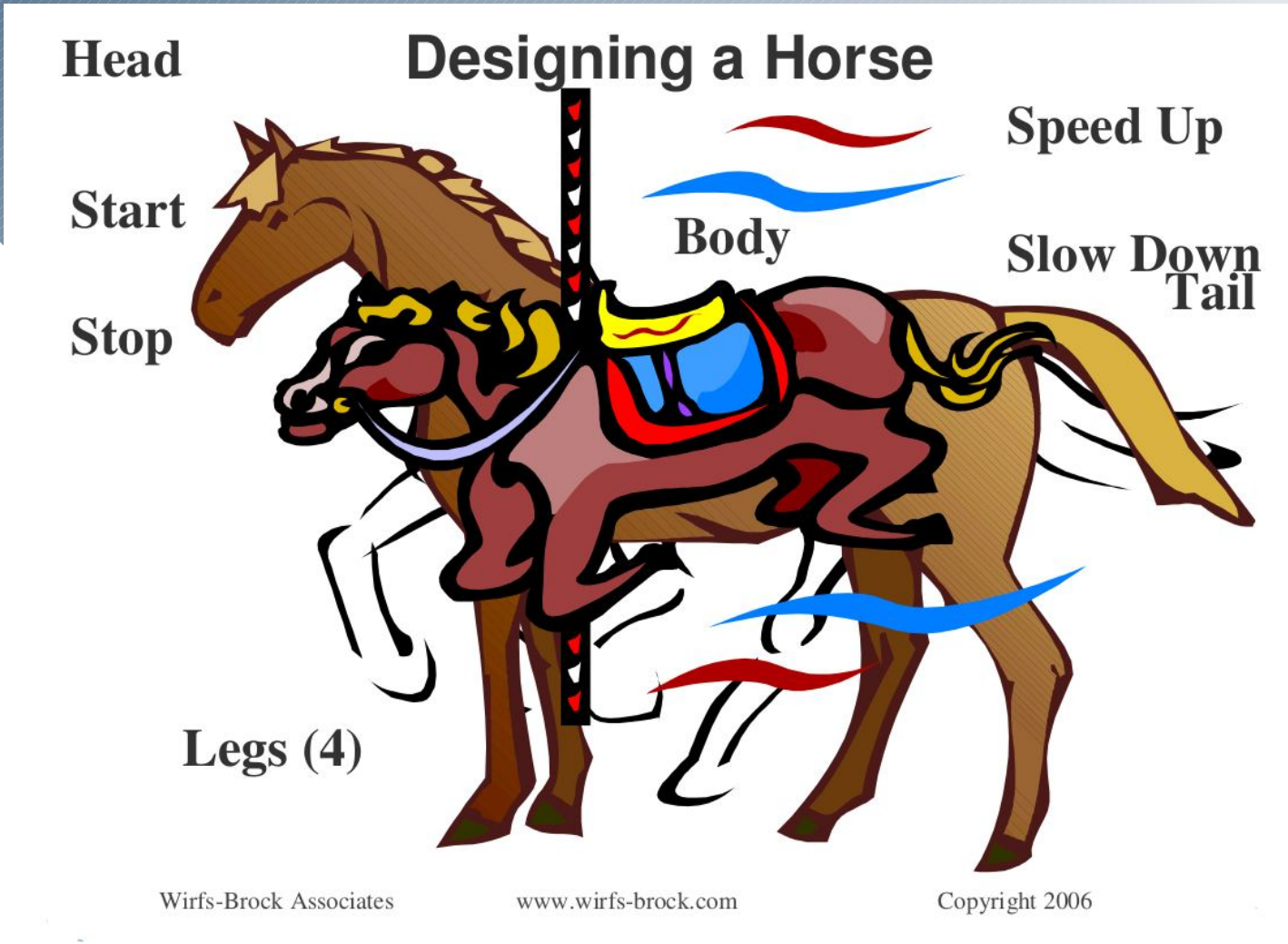
Example: designing a horse

- Objects: head, body, legs (4)
- Behavior: start, stop, speed-up, ...

Designing a horse responsibly:

- What is the horse for?
  - Riding
  - Competition
  - ...

OO Application Design
Responsibility-Driven Design



OO Application Design
RDD Principles

- **Maximize abstraction**

  Initially hide the distinction between data and behavior. Think of objects responsibilities for “knowing”, “doing”, and “deciding”. Who is the responsible for knowing this? Who is the responsible for doing this? A domain expert (maybe us) is needed.

- **Distribute behavior**

  Promote a delegated control architecture. Make objects smart— have them behave intelligently, not just hold bundles of data. Who is the responsible for printing a data in format MMDDYYYY?

- **Preserve Flexibility**

  Design objects so interior details can be readily changed, without affecting the rest of the system.

OO Application Design
Responsibility-Driven Design Constructs

- An **application** = a set of interacting objects
- An **object** = an implementation of one or more roles
- A **role** = a set of related responsibilities
- A **responsibility** = an obligation to perform a task or know information
- A **collaboration** = an interaction of objects or roles (or both)

OO Application Design
RDD: object modeling

- **Object modeling:**

  1. Identify object classes in the system
  2. Assign responsibilities to each object class

- A designer's story is a recommended tool for seeing what's important:

  Designer's story—a quickly written paragraph or two description of important ideas, what you know, and what you need to discover.

  - Identify story themes
  - Finding candidates
  - Identify **nouns** and **verbs** in the designer's story which may correspond to objects and their behavior

OO Application Design
Guidelines for Inventing Objects

- **Guidelines for Inventing Objects**
  - An object should capture one key abstraction in the problem domain
  - Choose meaningful names
  - Distinguish objects by behavior differences
  - Fit objects into their design context

OO Application Design
RDD: object modeling

Choose good NAMES for objects:

This is important!

- Name should match the role of the object in the system.
  E.g., a service provider performs work: StringTokenizer, ClassLoader, Authenticator.

- Name should give the programmer a clear idea of the responsibilities of the object (without extra details):
  TemporizadorConUnaPrecisionDeMasMenosDosMilisegundos →
  (Temporizador would be better)

- Use capitalization or underscores to separate words:
  LectorDeTarjetas or Lector_de_tarjetas
  - Be careful with abbreviations, as they may be confusing: TermProcess

OO Application Design
RDD: object modeling

Choose good NAMES:

- Avoid names with multiple interpretations:
  empty() : the object is empty or the method empties the object?

- Do not use digits

- Booleans: PrinterReady is better than PrinterState

- Java API may be a good source of inspiration for informative names.

OO Application Design
RDD: object modeling

Task: try to guess (without looking them up in the Java API) the purpose of these object types:

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

OO Application Design
RDD: object modeling

Keep a candidate when you can...

- Name it
- Define its purpose
- Assign it one or two initial responsibilities
- Understand how others view it

OO Application Design
RDD: assigning responsibilities

A responsibility is something a class knows or does.

E.g., a student may enroll in a course, take exams...

A single responsibility is larger than an operation or attribute; it usually involves more than one operation or attribute.

Use strong descriptions. The more explicit the action, the stronger the statement.

Stronger verbs: remove, merge, calculate, credit, activate

Weaker verbs: organize, record, process, maintain, accept

OO Application Design
RDD: assigning responsibilities

Guidelines for assigning responsibilities:

- Keep behavior with related information in the same object. This makes objects efficient.

- Don’t make any one role too big. This makes objects understandable.

- Distribute intelligence. This makes objects smart.

- Keep information about one thing in one place. This reduces complexity.

- An object can always do the work itself. A single responsibility can be implemented by one or more methods.

- Delegate part of a responsibility to one or more helper objects, whenever possible.

OO Application Design
RDD: collaboration model

What is a collaboration model?

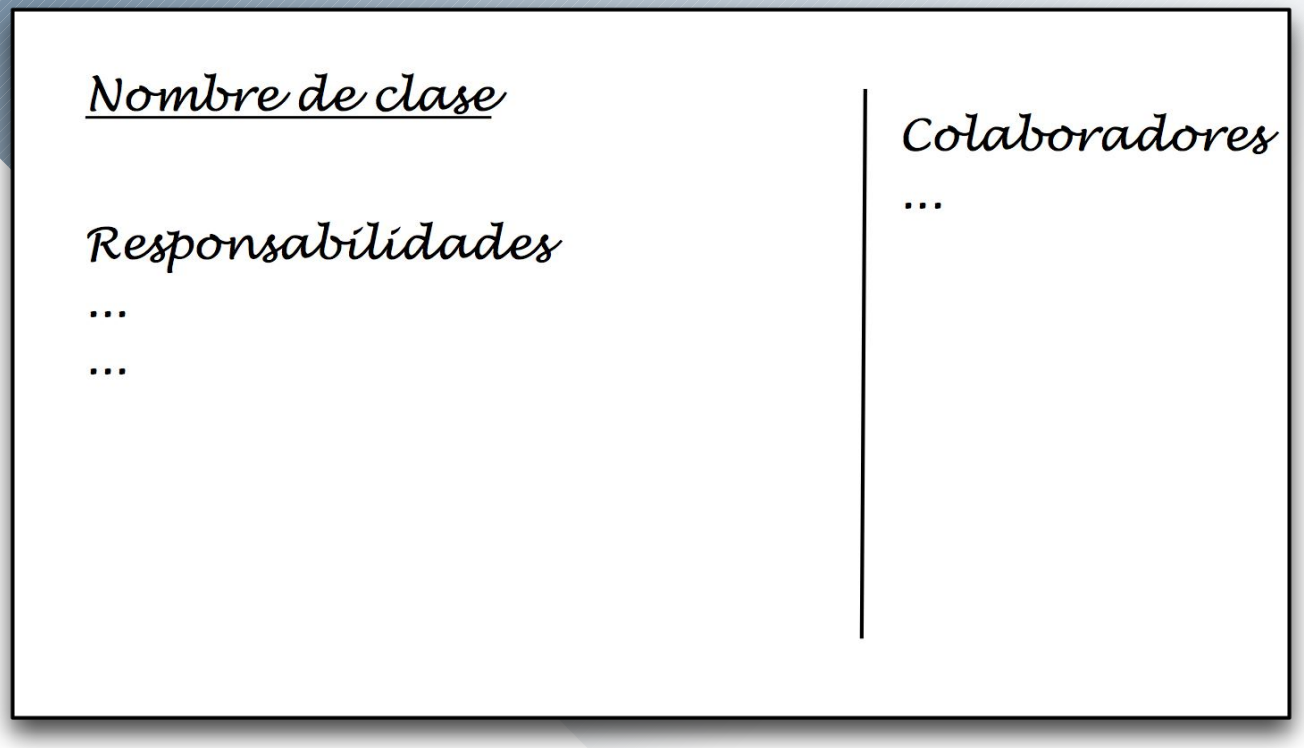
How a group of objects work together to fulfill a specific task.

It includes a description of objects, what each does, and how they interact.

We use CRC cards to record each object's responsibilities and collaborations.

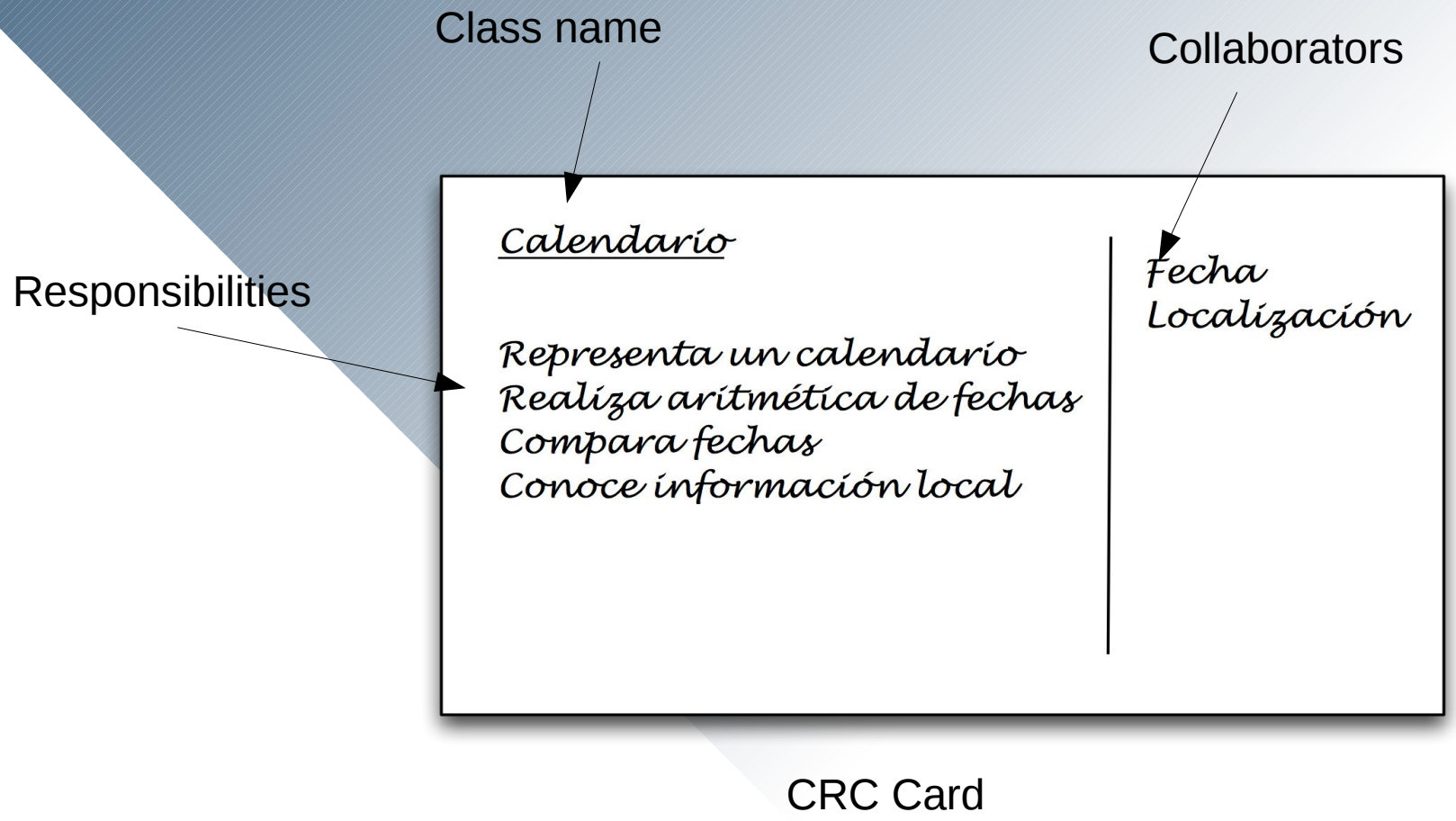
OO Application Design
CRC Cards

CRC: Class, Responsibility, Collaborators



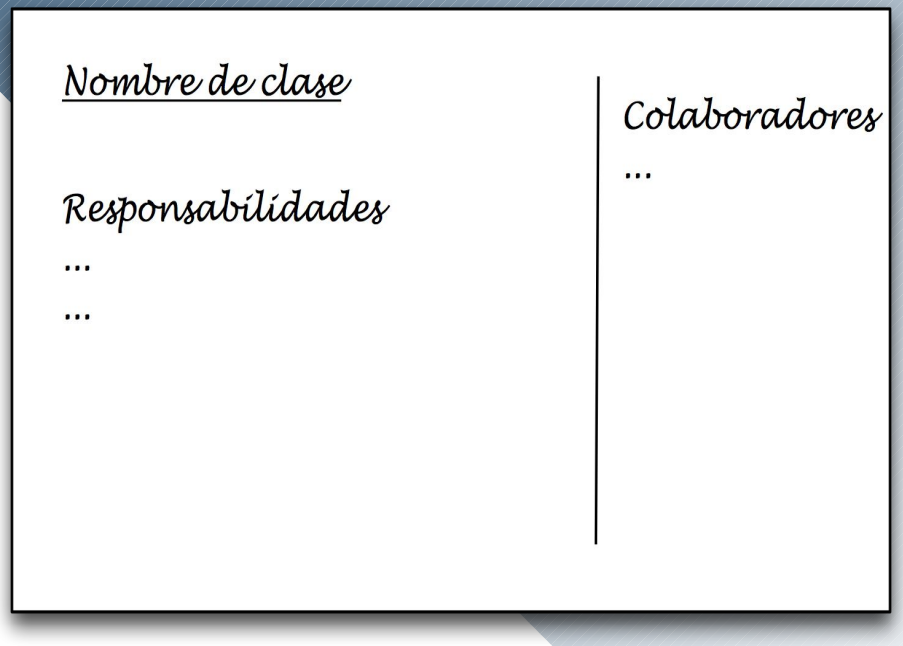
Components are represented using small index cards (the obsolete tool used to locate books in libraries before the 1990's).

OO Application Design
CRC Cards



CRC Card

OO Application Design
CRC Cards



Class name: it creates a vocabulary for discussing a design. It is important to find just the right set of words to describe our objects, a set that is internally consistent and evocative in the context of the larger design environment.

Responsibilities identify problems to be solved. A responsibility serves as a handle for discussing potential solutions. The responsibilities of an object are expressed by a handful of short verb phrases, each containing an active verb.

Collaborators. No object is an island: all objects stand in relationship to others, on whom they rely for services and control. We name as collaborators objects which will send or be sent messages in the course of satisfying responsibilities.

Object modeling with CRC cards

CRC cards promote high cohesion and low coupling. Card size may be a good approximation to the complexity of an object.

Cards can be...

- Stacked: parts below the whole
- Overlapped: implying close collaboration
- Layered: the most abstract card on top
- …

OO Application Design
CRC Cards

Object modeling with CRC cards

A very useful approach involves considering execution scenarios (what-if), where the team acts out the running of the application just as if it already possessed a working system.

1. **What must be done?**

2. **Who is going to do it?**

OO Application Design
CRC Cards: catalog sales example

Catalog sales system.

We want to develop a software application for a catalog sales system. The catalog contains a list of products with name, description, price and reference. Stock management is necessary to determine whether a particular product is available to meet the demand. Customers visit the shop, consult the catalog and indicate which products they want to buy and the number of items. If there is enough stock available for each product, a receipt containing the total price and shopping details is emitted and sent to the customer together with all the requested items.

OO Application Design
CRC Cards: ATM example

ATM (Automated Teller Machine)

We want to develop a software application to control an ATM (cash machine) which will only allow to make cash withdrawals with a charge card. Customers start using the ATM by introducing their card. After that, the system asks for a password, and, in case it is correct, it asks for the amount of money to withdraw (only multiples of 5 are allowed). Cards have a daily withdrawal limit; it the amount of money introduced by the customer does not exceed this limit, the software in the ATM checks whether the balance of the bank account associated to the card is high enough to permit the withdrawal. In case it is, the requested quantity is subtracted from the balance and given to the customer in 50, 20, 10 and/or 5 euros bills together with a receipt showing the withdrawn quantity, the remaining balance in the account and the card daily limit. Once the customer have taken away the money and the receipt, the card comes out. In case it takes more than one minute for the customer to take away the money, it would be recovered by the ATM and the operation would be canceled. In case it takes more than one minute for the customer to take away the card, it would be recovered by the ATM and held until a later customer claim.

Bibliography

- T. Budd. ***An Introduction to Object-oriented Programming, 3rd ed.***
  - Ch. 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