ECOOP'99 Workshop

Introducing OO Design and Programming - with Special Emphasis on Concrete Examples

Lisbon, June 1999

Designing and Coding Object-Oriented Systems: An Introductory Course Experience Report

Fernando Brito e Abreu
Miguel Goulão

INESC
Rua Alves Redol, 9, 1000 Lisboa, Portugal
email: {fba | miguel.goulao} @inesc.pt
phone: +351 1 3100263
fax: +351 1 3145843

Abstract

An emphasis on design issues along with introductory programming courses is fundamental for students to envision "the big picture". Teaching with the object-oriented paradigm from the start pays-off. An introductory course (CS/1 like), where this approach was adopted during three school years, is outlined. A detailed look on archetype problems solved at this course is presented.

1. Introduction

For the past decade we have witnessed a widespread tendency to concentrate the introductory CS/1-like courses on algorithmic programming, besides the natural effort on learning the details of language constructs. The intricacies of algorithms, like sort and search ones, and their expression in a given programming language are explored, often to the exhaustion, without concern for some at least as important aspects, such as design, reuse and programming in the large. In fact, that approach seems not to have changed considerably since the inception of the Computer Science curricula in the sixties.

This paper presents some lessons learned on shaping and teaching an undergraduate CS/1-like course, where a transition from a traditional procedural-oriented introductory programming curriculum to an object-oriented one, with emphasis on design, was pursued. This course, with an emphasis on design and reuse, was taught between 1996 and 1998 at the Superior Institute of Economics and Management of the Lisbon Technical University.

The following section will present the details of the above mentioned course, covering aspects as target group, preliminary knowledge, duration, objectives, topics covered, list of materials, formalisms and development platforms used. Section 3 deals with the identification of a set of problem archetypes and corresponding instances. Finally we draw some conclusions and identify future work in section 5.

2. Course description

This section includes some details about the course, namely its name, target group, preliminary knowledge, class and off-class expected time dedication, topics covered and list of materials given to students.

Course Title

Informatics 1

Target group

First year undergraduate students of "*Applied Mathematics to Economics and Management*". Around 60 students followed this course annually.

Preliminary knowledge

No preliminary knowledge on any kind of computer science topic was expected from the students. Experience showed that those who acquired, in high school, some programming experience with procedural languages such as *BASIC*, often suffered some paradigm shift symptoms, that slowed down their learning process at the earlier course stages. On the other hand, they had less difficulty on understanding the algorithmic-related topics.

Duration issues

This course was taught during the first semester, which duration is approximately 18 to 20 weeks. In each week three classes took place: two lectures and one lab, on distinct days. All classes lasted for 1,5 hours. Lecture classes had around 30 students (two shifts), while laboratory classes had groups of 15 students (4 shifts). Regular assignments were such that students were expected to spend an additional period of three to four hours of homework, each week. A more complete team assignment was given to students by the end of the term, which implied some extra dedication during around two weeks.

Objectives

- Brief understanding of the internal structure, capabilities and limitations of a digital computer and its cost/benefit evaluation;
- development of abstraction capabilities in order to produce specifications, particularly in domains of Applied Mathematics, using the object-oriented paradigm;
- knowledge of the techniques needed to reify the specified problems, namely those allowed by the available computational tools;
- capability of reusing and integrating ready-made components.

Topics covered

Information storage and internal representation:

- binary and hexadecimal systems
- conversion between numbering systems
- character encoding
- storage devices

Internal architecture and interconnection of computers and peripherals

Introduction to operating systems

Syntax and semantics of programming languages:

- syntactic diagrams
- BNF notation

Generic concepts about object-oriented programming:

- classes and objects
- methods and instance variables
- encapsulation and information hiding
- message exchange
- inheritance and polymorphism

Generic concepts about object-oriented design:
- generalization and specialization
- relationships between classes
- composition and reuse
- modularization criteria

Modeling in OMT:
- representation of classes, methods and instance variables
- representation of inheritance, association and aggregation
- Conversion of OMT to source code

Control structures

Standardization of coding style

Introduction to a development environment:
- separated compilation, linkage and execution
- libraries and their use
- interactive context-sensitive hypertext help systems

Building object systems:
- representation and manipulation of polynomials
- representation and manipulation of vectors and matrixes
- representation and manipulation of complex numbers
- graphical representation of functions in R^2 and R^3.

List of materials

A large set of materials was used in the realm of this course. Those included:
- set of fully commented overhead transparencies covering most aspects of the proposed course curriculum [Abreu98];
- class diagram of a package containing graphical widgets such as 2D shapes, buttons, text boxes, scroll bars, etc.
- graphical package in linkable format;
- large set of simple coding examples using the Graphical Package;
- class diagram and source code of a full-featured case study used in classroom – a tool for visualizing R^3 equations in several perspectives;
- set of exams and corresponding solutions.

All these materials were made available to students in paper or electronic version (for download in the Internet).

Formalisms used

We used the Object Modeling Technique (OMT) proposed in [Rumbaugh91] for design and an object-oriented extension of Pascal as included in Borland's Delphi environment [Borland95, Cantù96], for programming purposes.

Development platforms

Design examples handed out to students were produced with the Paradigm Plus tool, a Platinum Software (http://www.platinum.com) product. The adopted coding platform was Borland's Delphi (http://www.borland.com).
The Graphical Package we used was originally developed at the Computer Science Department at Brown University (USA), for a now out-dated environment, Borland's TurboPascal. That MS-DOS interface environment subsequently evolved to the Windows interface environment called Delphi. The mentioned package was built with an extended version of an OO Pascal language not fully supported by Delphi, namely in what concerned the object model. We updated and extended the Graphical Package to be able to work in recent versions of Delphi.

3. Archetype problems

The expected outcome of a given engineering course, such as the one described here, is the student's ability to solve a new set of problems. We tried to characterize the type of problems in the following archetypes:

- ***Archetype 1 – From requirements specification to design***

This problem archetype refers to the issue of producing a design model from a requirements specification. In the context of this course, the specification was expressed in natural language and only the OMT Object Diagram of the design model is required.

- ***Archetype 2 – Understanding a design (requirements identification)***

This archetype is the reverse of archetype 1. The objective here is to obtain a problem description out of an existing design model. A successful completion of this activity means the student is able to understand both the design notation and the problem domain being described. The problem solution is a requirements' specification in natural language.

- ***Archetype 3 – From design to implementation***

The kind of problem characterized by this archetype corresponds to the generation of an implementation using a given OO programming language, out of a given design model. Students must be knowledgeable in the translation rules that apply.

- ***Archetype 4 – Application building***

This problem archetype corresponds to the ability of producing (simple) applications in source code format, from a given requirements specification (expressed in natural language) and constrained by the use of a given class structure (expressed by means of an OMT Object Diagram).

- ***Archetype 5 – Design capture***

This last archetype refers to the reverse engineering problem of a given implementation. A source code extract is made available for interpretation and students are supposed to express the underlying Object Diagram.

For each of the archetypes, we now give an instance, along with a corresponding solution.

4. Archetype instances

Archetype 1 - From requirements specification to design

Problem statement:

Model in OMT (Object Modeling Technique) the following universe of discourse, by only representing classes, their attributes, relations (inheritances, associations and aggregations) and corresponding cardinality.

"***The Portuguese Playball Federation*** decided to develop a computerized system to manage information about the matches, players, referees, teams, stadiums, and disciplinary punishments. Each season, characterized by the year, has a set of rounds, numbered sequentially. Within each of those rounds, a set of matches is played. Each match is played on a given date and has an assigned referee (of whom we know the name, address and referee course finishing date). The home and away teams play each match at a stadium, which has a name and a maximum attendance capacity and may, or may not, belong to a team. Each team has a name and an office. Each playball player has a name, address, and birth date and is inscribed to play for a given team. Some of the playball players are professionals and those have a sport license. The system should hold information about who were the players in each match and, conversely, what matches each player participated in. Sometimes, players are punished during a match by a referee. The federation wishes to keep a description of the events that led to each punishment, as well as information on the corresponding player, referee and match where it took place."
Hint: use the class Date whenever appropriate.

Problem solution:

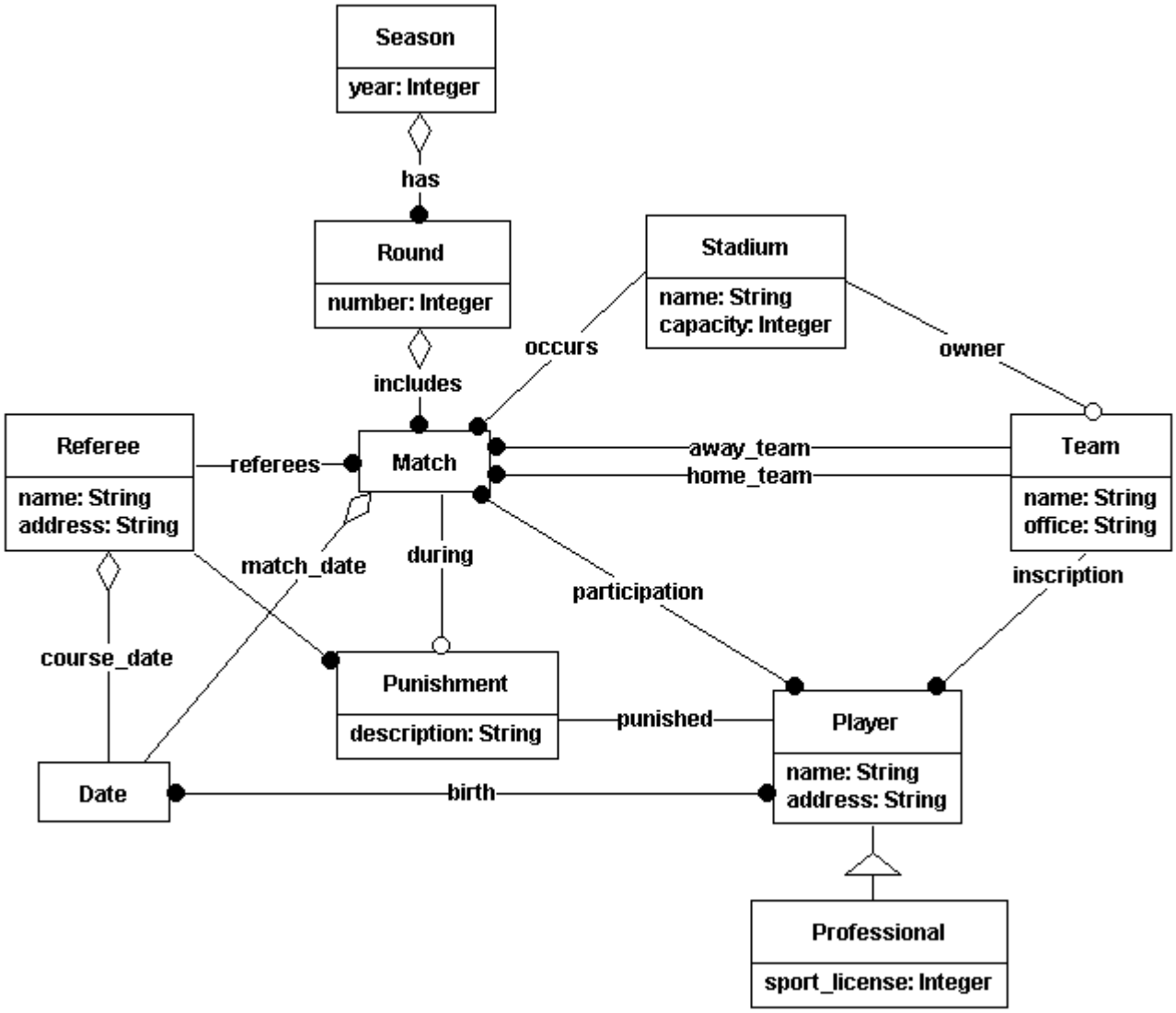

Diagram 1 – Fragment of PlayBall Federation information system

Archetype 2 – Understanding a design (requirements identification)

Problem statement:

The following OMT diagram represents a fragment of ***Air Lusitânia's*** information system. This system manages information about the staff, the company airplanes, flights, tickets sold and passengers that flew with company. Some of the latter have subscribed a frequent flyer program.

Write a natural language description that could have generated this diagram:

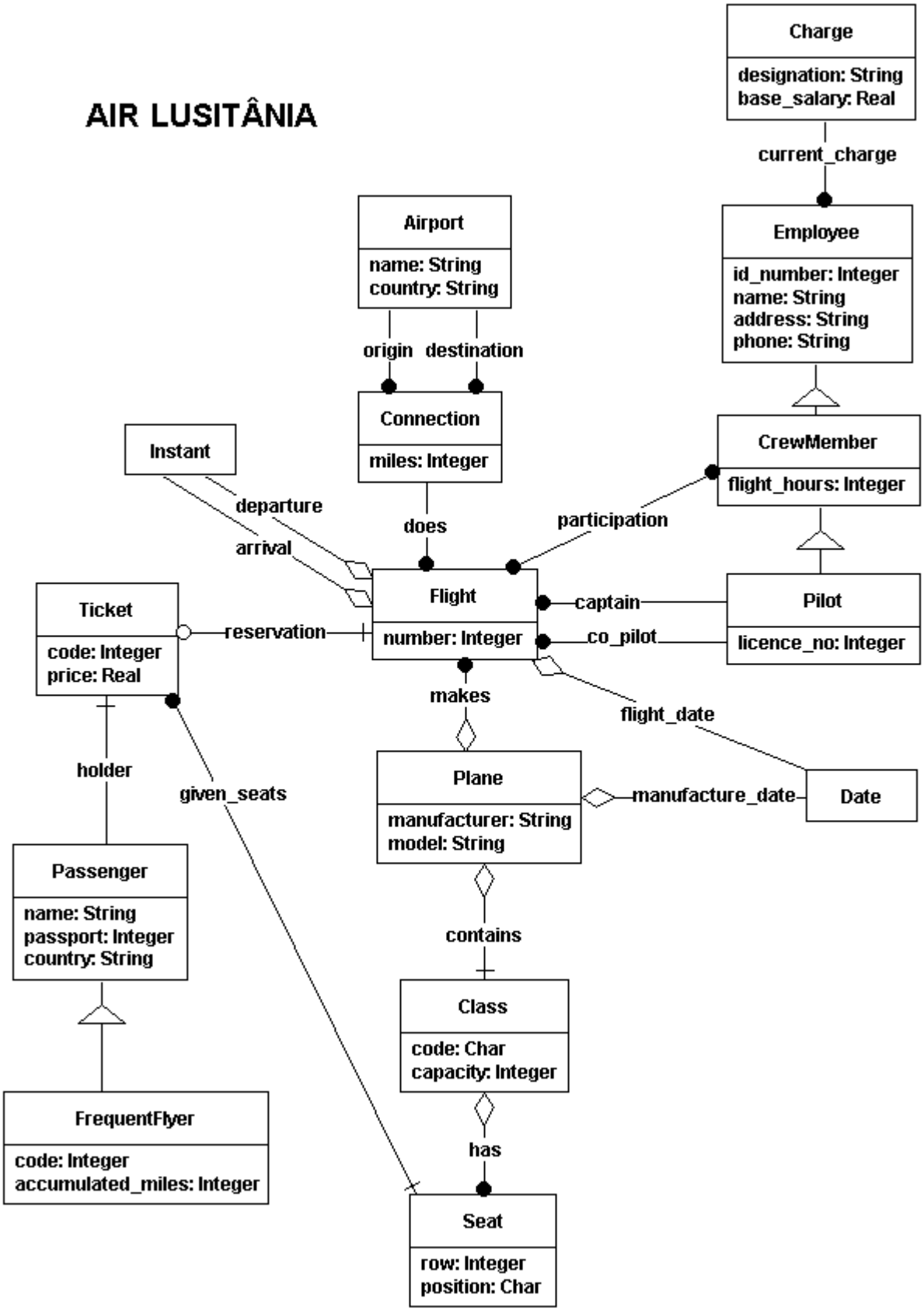

Diagram 2 – Fragment of Air Lusitânia information system

Problem solution:

All ***Air Lusitânia*** employees have an id number and information stored about their name, address, phone number and current charge. Each charge has a designation and a base salary. Several employees can have the same charge. Some of those employees are crewmembers, and for those we also need to know which is the number of hours they flew. For the special case of pilots, we still need their license number (brevet).
About each plane we know their manufacturer, manufacture date and model. Each one contains a given number of classes (at least one) identified by a code and having a fixed capacity. Each seat within a class has a row (1, 2, 3,...) and position (A, B, C,...) identifier. A plane makes flights, which are numbered sequentially. Each flight, executed in a given date and with departure and arrival instants known, corresponds to a connection between two airports (origin and destination). We know the distance in miles for each connection. Each airport has a name and is situated in a given country. Two pilots are assigned (captain and co-pilot) to each flight, along with a set of other crewmembers.
Tickets issued are tagged with a code, their price is recorded and they have a holder (a passenger). They can contain the reservations of one or more flights. Upon check-in a ticket is also given one or more seats (one per each flight). Passenger name, passport number and country of origin are recorded. Some of those passengers have subscribed the ALF (Air Lusitânia Frequent-flyer) Program. For those, identified by a code number, the number of accumulated miles flown is recorded.

Archetype 3 – From design to implementation

Problem statement:

Consider the system in Diagram 2 – Fragment of Air Lusitânia information system.

a) Produce an implementation of the class interfaces in OOPascal. Suppose that flights made by each plane, classes in each plane, seats in each class, reserved flights, seats given for a ticket and crew members that participated in a given flight are represented, respectively, by the following data structures:

Plane.makes: ARRAY[1..MaxFlights] OF Flight;
Plane.contains: ARRAY[1..MaxClasses] OF PlaneClass;
PlaneClass.has: ARRAY[1..MaxSeats] OF Seat;
Ticket.reservation: ARRAY[1..MaxReservations] OF Flight;
Ticket.given_seats: ARRAY[1..MaxReservations] OF Seat;
Flight.participation: ARRAY[1..MaxCrew] OF CrewMember;
{MaxFlights, MaxClasses, MaxSeats, MaxReservations and MaxCrew are predefined constants}

Create operations (constructors, destructors, selectors, etc) in those interfaces as needed to solve item b)

b) To plan maintenance actions we need to know the total number of hours flown by the plane and the total number of miles it has done. For that purpose, implement the two following operations:

FUNCTION Plane.TotalMilesFlown: Integer;
FUNCTION Plane.TotalHoursFlown: Integer;

Hint: Use the operation Instant.TimeElapsed(since: Instant): Real for obtaining the flight periods.

Problem solution a):

```
Passenger =CLASS
        PRIVATE
          name: String;
          passport: Integer;
          country: String;
        PUBLIC
          CONSTRUCTOR New;
          DESTRUCTOR Destroys;
          PROCEDURE Reads;
       END;

FrequentFlyer=
```

```
        CLASS (Passenger)
          PRIVATE
            code: Integer;
            accumulated_miles: Integer;
          PUBLIC
            PROCEDURE IncreaseMiles(new_miles: Integer);
        END;

Charge=    CLASS
          PRIVATE
            designation: String;
            base_salary: Real;
        END;

Employee=  CLASS
          PRIVATE
            id_number: Integer;
            name: String;
            address: String;
            phone: String;
            current_charge: Charge;
        END;

CrewMember=CLASS (Employee)
          PRIVATE
            flight_hours: Integer;
        END;

Pilot=     CLASS (CrewMember)
          PRIVATE
            licence_no: Integer;
        END;

Airport=   CLASS
          PRIVATE
            name: String;
            country: String;
        END;

Connection=CLASS
          PRIVATE
            miles: Integer;
            origin, destination: Airport;
          PUBLIC
            FUNCTION GetMiles: Integer;
            {selector of "miles" attribute}
        END;

Flight=    CLASS
          PRIVATE
            number: Integer;
            does: Connection;
            captain, co_pilot: Pilot;
            participation: ARRAY[1..MaxCrew] OF CrewMember;
            departure, arrival: Instant;
            flight_date: Date;
          PUBLIC
            FUNCTION GetConnection: Connection;
            FUNCTION GetArrival: Instant;
            FUNCTION GetDeparture: Instant;
```

```
    {selectors of "does", "arrival" and "departure" attributes}
  END;

Seat=    CLASS
  PRIVATE
    row: Integer;
    position: Char;
  END;

{Here we use the identifier PlaneClass instead of just Class like in the design, because the latter is a reserved keyword}

PlaneClass=CLASS
  PRIVATE
    code: Char;
    capacity: Integer;
    has: ARRAY[1..MaxSeats] OF Seat;
  END;

Plane=   CLASS
  PRIVATE
    manufacturer: String;
    model: String;
    manufacture_date: Date;
    contains: ARRAY[1..MaxClasses] OF PlaneClass;
    makes: ARRAY[1..MaxFlights] OF Flight;
  PUBLIC
    FUNCTION TotalMilesFlown: Integer;
    FUNCTION TotalHoursFlown: Integer;
  END;

Ticket=   CLASS
  PRIVATE
    code: Integer;
    price: Real;
    holder: Passenger;
    reservation: ARRAY[1..MaxReservations] OF Flight;
    given_seats: ARRAY[1..MaxReservations] OF Seat;
  PUBLIC
    CONSTRUCTOR New;
    DESTRUCTOR  Destroys;
    PROCEDURE SetHolder(a_passenger: Passenger);
    PROCEDURE CalculatesPrice;
    PROCEDURE CreatesCode;
    PROCEDURE SetReservation(number: Integer; the_flight: Flight);
    FUNCTION  GetReservation(number: Integer): Flight;
    PROCEDURE SetSeat(number: Integer; the_seat: Seat);
    FUNCTION  GetSeat(number: Integer): Seat;
  END;
```

Problem solution b):

```
FUNCTION Plane.TotalMilesFlown: Integer;
VAR total_miles, i: Integer;
BEGIN
 total_miles:= 0;
 i:= 1;
 WHILE (i<=MaxFlights) AND (makes[i] <> NIL) DO
   BEGIN
```

```
    total_miles:= total_miles + makes[i].GetConnection.GetMiles;
    i:= succ(i)
   END;
  TotalMilesFlown:= total_miles
END;

FUNCTION Plane.TotalHoursFlown: Integer;
VAR
  i: Integer;
  total_hours: Real;
BEGIN
  total_hours:= 0;
  i:= 1;
  WHILE (i<=MaxFlights) AND (makes[i] <> NIL) DO
    BEGIN
      total_hours:= total_hours +
        makes[i].GetArrival.TimeElapsed(makes[i].GetDeparture);
      i:= succ(i)
    END;
  TotalHoursFlown:= Round(total_hours)
END;
```

Archetype 4 – Application building

Problem statement:

Based on the class structure presented in Diagram 2 – Fragment of Air Lusitânia information system, write a small application with the following options:

- Ticket emission (with creation of holder if it does not exist) with recording of reserved flights;
- Seat booking for the flights of a given ticket (input: ticket code);
- Update of frequent flyer miles (given ticket and frequent flyer codes);
- Program exit.

Consider that, for all classes, the basic constructor is called **New,** destructor is called **Destroys** and basic input and output operations are called **Reads** and **Writes**. Also consider the existence of the Database class, with the following interface:

```
Database =
      CLASS
        PUBLIC
          CONSTRUCTOR New;
          DESTRUCTOR  Destroys;
          FUNCTION SeekEmployee(employee_id: Integer): Employee;
          FUNCTION SelectCharge: Charge;
          FUNCTION SelectConnection: Connection;
          FUNCTION SelectFlight(one_connection: Connection): Flight;
          FUNCTION SelectSeat(flight_no: Flight): Seat;
          FUNCTION SeekTicket(ticket_code: Integer): Ticket;
          FUNCTION SeekPassenger(passport_no: Integer): Passenger;
          FUNCTION SeekFrequentFlyer(code: Integer): FrequentFlyer;
        END;
```

All those operations return NIL when the requested information is not found (seek operations) or the user gives up (select ones):

Problem solution:

```
PROGRAM AirLusitania;
```

```
USES
  Lusitania; {module containing class definitions}

VAR
  i, option, a_passport, a_code, num_flights: Integer;
  a_database: Database;
  a_passenger: Passenger;
  a_frequent: FrequentFlyer;
  a_ticket: Ticket;

BEGIN
  a_passenger:= Passenger.New;
  a_ticket:= Ticket.New;
  a_database:= a_database.New;
  {Establishes connection with existing a_database}

  REPEAT
    REPEAT
      Writeln('0 - Exit program');
      Writeln('1 - Ticket emission');
      Writeln('2 - Check-in (Seat reservation for a given ticket)');
      Writeln('3 - Update frequent flyer miles');
      Writeln;
      Write('Option > ');
      Readln(option)
    UNTIL option IN [0, 1, 2, 3];

    CASE option OF
      1: BEGIN   { Ticket emission }
           Write('Passport number >');
           Readln(a_passport);
           a_passenger:= a_database.SeekPassenger(a_passport);
           IF a_passenger = NIL THEN
             a_passenger.Reads;

           a_ticket.SetHolder(a_passenger);
           Write('Flight numbers to include in the ticket > ');
           Readln(num_flights);
           FOR i:= 1 TO num_flights DO
             a_ticket.SetReservation(i, a_database.SelectFlight(a_database.SelectConnection));
           a_ticket.CalculatesPrice;
           a_ticket.CreatesCode;
         END;

      2: BEGIN    { Check-in }
           Write('Ticket code >');
           Readln(a_code);
           a_ticket:= a_database.SeekTicket(a_code);
           i:=1;
           WHILE (i<=MaxReservations) AND (a_ticket.GetReservation(i) <> NIL) DO
             BEGIN
               a_ticket.SetSeat(i, a_database.SelectSeat(a_ticket.GetReservation(i)));
               i:= i+1
             END
         END;

      3: BEGIN    { Update Frequent Flyer miles }
           Write('Frequent Flyer Code>');
           Readln(a_code);
           a_frequent:= a_database.SeekFrequentFlyer(a_code);
```

```
        IF a_frequent <> NIL THEN
          BEGIN
            Write('Ticket number >');
            Readln(a_code);
            a_ticket:= a_database.SeekTicket(a_code);
            i:=1;
            WHILE (i<=MaxReservations) AND (a_ticket.GetReservation(i)<> NIL) DO
              BEGIN
                a_frequent.IncreaseMiles(a_ticket.GetReservation(i).GetConnection.GetMiles);
                i:= i+1
              END
          END
      END;
  UNTIL option = 0;

  a_passenger.Destroys;
  a_ticket.Destroys;
  a_database.Destroys;

END.
```

Archetype 5 – Design capture

Problem statement:

Consider the following code extract and produce an OMT Object Diagram that might have originated it.
Hint: use link associations if appropriate.

```
TYPE

Category = (Lecturer, AuxiliarProf, AssociateProf, FullProf);
Degree = (BSC, MSC, PHD);

Date = String;

Department =
        CLASS
          PUBLIC
            CONSTRUCTOR Init;
            DESTRUCTOR Terminate;
            PROCEDURE StaffAdmission;
            PROCEDURE ElectPresident(new_president: Staff);
            PROCEDURE Write;
          PRIVATE
            name: String;
            president: Staff;
        END;

Person =  CLASS
          PUBLIC
            CONSTRUCTOR Init;
            DESTRUCTOR Terminate;
            PROCEDURE Write;
          PROTECTED
            number: Integer;
            name: String;
            address: String;
```

```
        END;

Staff =   CLASS(Person)
          PUBLIC
            CONSTRUCTOR Init(joined_dep: Department);
            DESTRUCTOR Terminate;
            PROCEDURE Write;
          PRIVATE
                        contract: Department;
            contract_date: Date;
            staff_Category: Category;
        END;

Student = CLASS(Person)
          PUBLIC
            CONSTRUCTOR Init;
            DESTRUCTOR Terminate;
            PROCEDURE Write;
          PROTECTED
            matriculation_date: Date;
            matriculation_grade: Integer;
        END;

Course =  CLASS
          PUBLIC
            CONSTRUCTOR Init(the_chair: Staff);
            DESTRUCTOR Terminate;
            PROCEDURE Write;
          PRIVATE
            name: String;
            chair : Staff;
        END;

Specialization =
        CLASS
          PUBLIC
            CONSTRUCTOR Init;
            DESTRUCTOR Terminate;
            PROCEDURE Enroll;
            PROCEDURE Write;
          PRIVATE
            name: String;
            spec_degree: Degree;
            numerus_clausus: Integer;
               END;

Enrollment =
        CLASS
          PUBLIC
            CONSTRUCTOR Init(whichStudent:Student;
                          whichCourse:Course);
            DESTRUCTOR Terminate;
            PROCEDURE Write;
          PRIVATE
            enrollment_date: Date;
            enrollment_number: Integer;
            grade_obtained: Integer;
            studentEnrollment: Student;
            courseEnrollment: Course;
        END;
```

Problem solution:

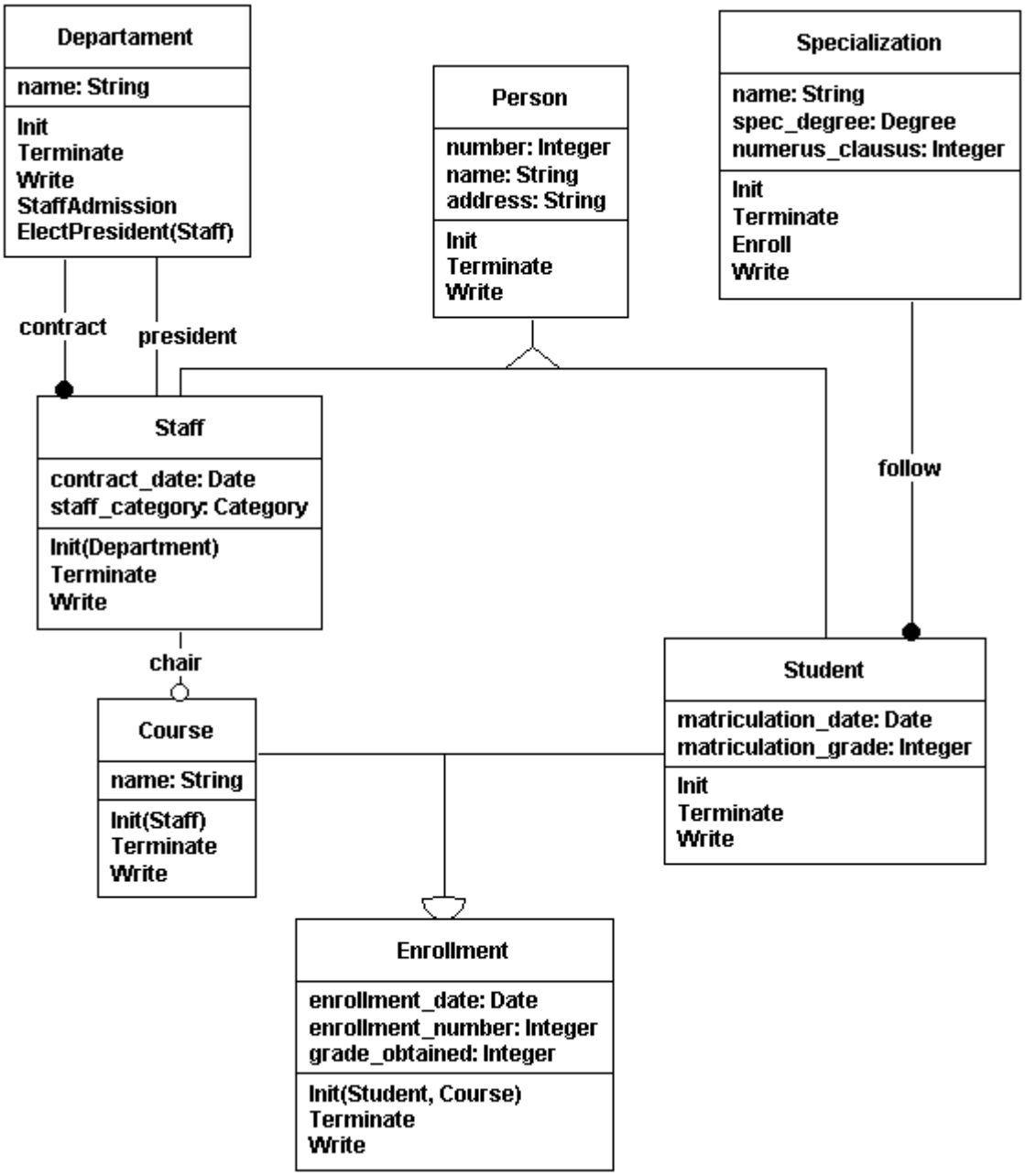

Diagram 3 – Fragment of an university information system

5. Conclusions and future work

We think design issues should be more emphasized from the start. The use of the object-oriented (OO) paradigm, where the conceptual gap between design and programming is not as large as in the procedural paradigm, makes the job easier. Our students were more motivated than in previous years by being able to use diagrammatic techniques that allowed them to model the real world and make their case studies more tangible. More complex case studies and team assignments were understood and accomplished by the students.

We plan to progress in three directions. First, we plan to cooperate with professors in charge of the other courses that the students are attending simultaneously with our own courses. Bridging the teaching curriculum of often-divorced courses will surely increase the students' motivation and strengthen their ability to build a problem-solving mind. We believe that a careful choice of requirements specifications taken out of the problems dealt with in those other courses, will increase the cross-fertilization of ideas and the awareness of the potential of using computing power to solve complex or data-intensive problems.

Second, we want students to be aware of the body of knowledge on designing OO systems, which was condensed in the last years – the so-called Design Patterns [Gamma95]. In an introductory course we do not expect students to apply them, but simply to be able to understand them.

Last, but not the least, we think it is important for the students to use the result of the standardization effort on OO design notations – the Unified Modeling Language [Booch97], which seems to be here to stay. We plan to adopt it in the next course release.

Complementary to the curricular issues dealt with in this paper, are the ones on pedagogy, which have not been addressed here. We expect that the undergoing work in Pedagogical Patterns, that follows early proposals as [Lilly96, Abreu97], will bring new insights to help teachers to fulfill their objectives.

References

[Abreu97] Abreu, Fernando Brito, "Pedagogical Patterns: Picking Up the Design Patterns Approach", *Object Expert*, Vol. 2, n. 3, SIGS (UK), March/April 1997.

[Abreu98] Abreu, Fernando Brito; Goulão, Miguel; *Sebenta de Informática I (MAEG)*, ISEG, 1998. *(in Portuguese)*

[Booch97] Booch, G. & Jacobson, I. & Rumbaugh, J., "UML Semantics," Rational Software Corporation, Version 1.0, January 1997.

[Borland95] Borland, *Delphi 2 - User Manual*, Borland Corporation, 1995.

[Cantù96] Cantù, Marco; *Mastering Delphi 2 for Windows 95/NT*, Sybex, 1996. (ISBN: 0-7821-1860-7)

[Conner95] Conner, B.; D. Niguidula; A. van Dam; *Object-Oriented Programming in Pascal - a Graphical Approach*, Addison-Wesley, 1995. (ISBN:0-7821-1860-7)

[Gamma95] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[Griesel94] Griesel, Martha Ann; Welz, Linda, "Using Modified Fagan Inspections to Control Rapid System Development", *Proceedings of the 4th International Conference on Software Quality*, McLean, Virginia, USA, October 1994.

[Lilly96] Lilly, Susan, "Patterns for Pedagogy", *Object Magazine*, January 1996, pp. 93-96.

[Rumbaugh91] Rumbaugh, J.; Blaha, M.; Premerlani, W. Eddy; F., Lorenson, W., Object-*Oriented Modelling and Design*, Prentice Hall, 1991.