Practical sheet #05

5.1 Objectives

1. Practise using **Class Diagrams** and **Sequence Diagrams**;
2. Relate Class and Sequence diagrams to the implementation they represent.
3. Develop the ability to use Class and Sequence diagrams to design systems.

5.2 Exercises

Solve the exercises below by creating the diagrams requested.

5.2.1 Online shopping

Consider the Java code extract shown in Annex A.

Analyse the code shown and, knowing that the models should describe the logic of the solution and not necessarily all the Java code that has been written, develop the following diagrams:

1. A **Class Diagram** that describes the architecture present in the code.
2. A **Sequence Diagram** describing the behaviour of the method `comprados(String bi)` — the method calculates a list of all the tickets bought by a given buyer.

5.2.2 Assignments Evaluation System

Consider the Java code excerpt presented in Appendix B, relating to a subsystem for managing practical assignments at a university.

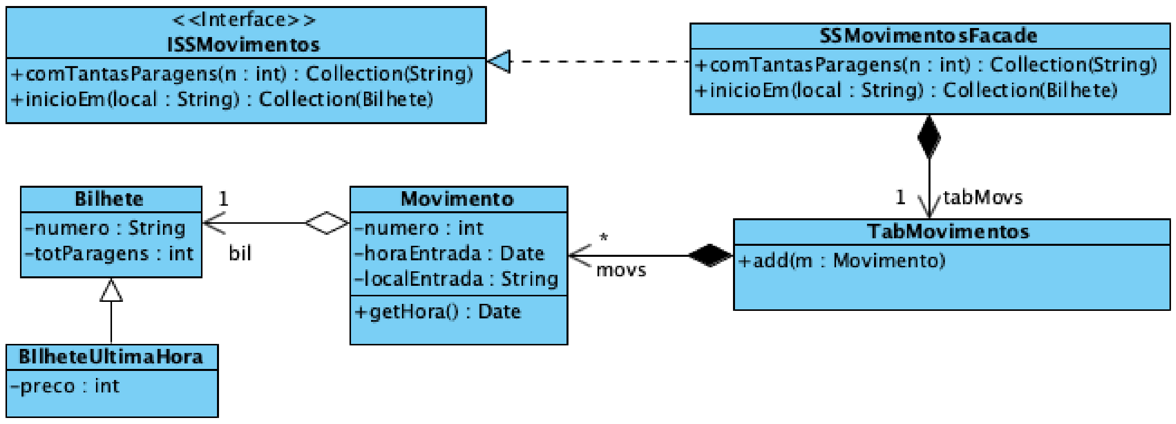


Figure 5.1: Class diagram for the Exercise 5.2.3

In relation to the code presented, solve the following exercises:

1. Analyse the code and present the corresponding **Class Diagram**, trying to be as exhaustive as possible in identifying the relationships between the classes. Consider that at the *Facade* level, all associations correspond to compositions.

2. Draw **Sequence Diagrams** for the following methods:

   (a) `public Student getStudent(String codStudent)` — The method should return the student with the indicated number.

   (b) `public int getStudentMark(String codStudent)` — The method should calculate a student's mark, knowing that the theory and practical marks are worth 60% and 40% of the final mark respectively.

   (c) `public void registerDelivery(Delivery e, String codGroup)` — The method should register a delivery in the group indicated, if there is not already a delivery for that date.

   (d) `public boolean validateAssessors()` — The method should check that no student is an assessor for their own group.

5.2.3 Transport Company

Consider the class diagram shown in Figure 5.1, which represents a solution for a public transport company, and answer the following questions:

1. Define a precondition for the operation `add(Movimento)` (see `TabMovimentos`) to ensure that the operation does not violate the following constraint:

   There cannot be two moves with the same number in an instance of TabMoves.

2. Write a **Sequence Diagram** for the operation:

   ```
   comTantasParagens(n: int): Collection(String)
   ```

   of class `SSMovimentosFacade`, which determines the codes of all tickets that made journeys (`Movimentos`) with a number of stops equal to the value `n` given as a parameter.

3. Write a **Sequence Diagram** for the operation:

   ```
   inicioEm(local: String): Collection(Bilhete)
   ```

   of class `SSMovimentosFacade`, which determines all the tickets that made journeys starting at `local`.

5.2.4 Car Parks

Consider that, in the context of the University Information System, you want to model a car park administration sub-system. In this system, in order to be able to park in the various car parks, users must have an identifier that is associated with a particular vehicle. If a customer has more than one car, they need to acquire as many identifiers as they have cars. Whenever an identifier is detected in a particular car park, the time and date of entry and the time and date of exit from the car park are recorded. Each car park has a price list based on the type of vehicle. This information is provided by the association between the identifier and the vehicle.

The analysis team and the client jointly developed the domain model shown in Figure 5.2. Knowing that during the requirements analysis it was defined that it should be possible for a user to search for and list their movements in the various parks, as well as to obtain monthly account statements for a given identifier. statements for a given identifier. Based on the Use Case, it was possible to identify the need to implement the operations mentioned in (3) to (5) below and, finally, knowing that:

- the direction to be used in the associations must be defined according to the methods requested in (3) and (4);
- to simplify, in a first approach, the park management subsystem has as *Facade* the class `GesParkFacade` which implements the interface of Figure 5.3.

Answer the following questions:

1. Construct the **Class Diagram** of the parks management sub-system architecture, based on the information provided in the Domain Model and knowing

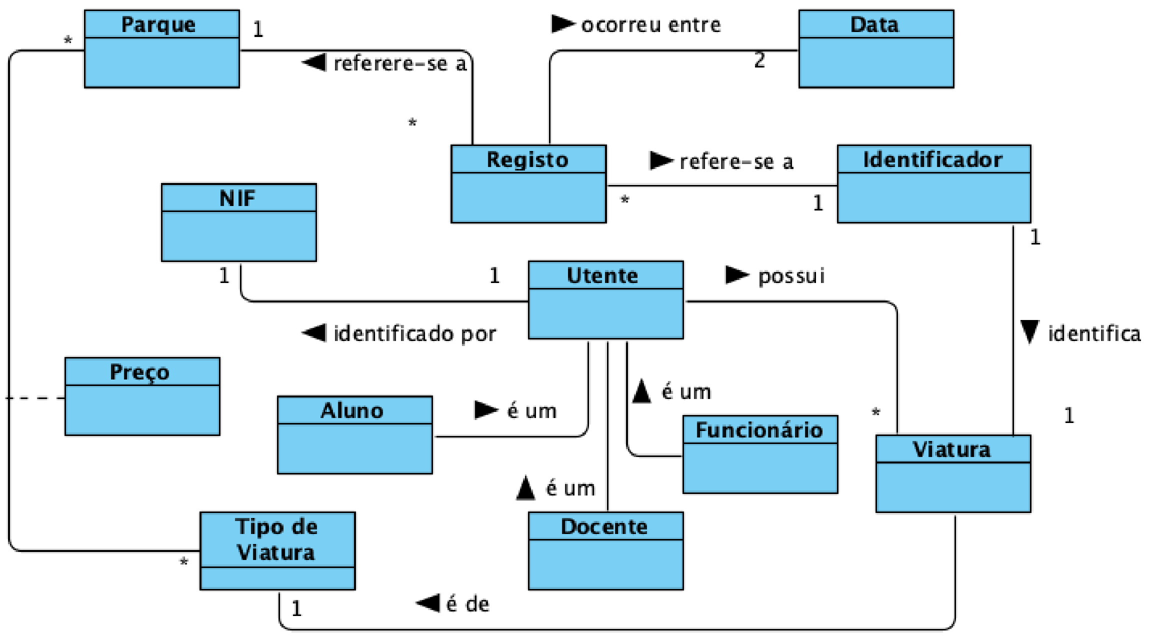
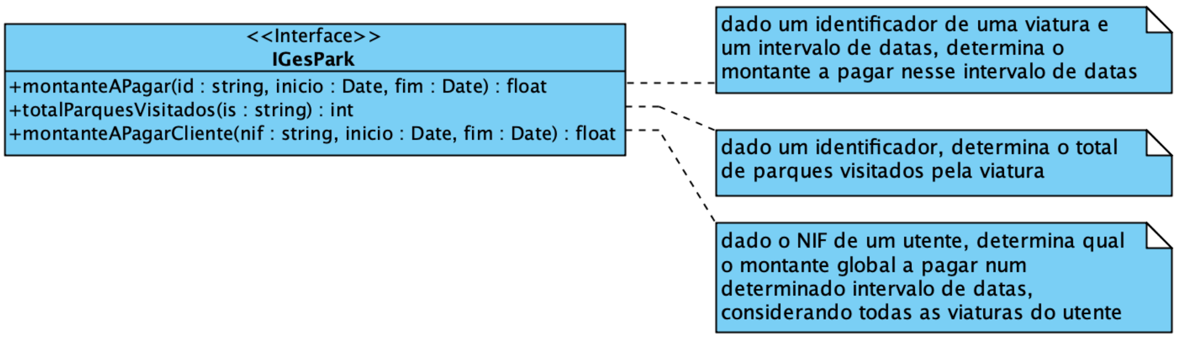


Figure 5.2: Domain Model for the Exercise 5.2.4



```
<<Interface>>
IGesPark
+montanteAPagar(id : string, inicio : Date, fim : Date) : float
+totalParquesVisitados(is : string) : int
+montanteAPagarCliente(nif : string, inicio : Date, fim : Date) : float
```

dado um identificador de uma viatura e um intervalo de datas, determina o montante a pagar nesse intervalo de datas

dado um identificador, determina o total de parques visitados pela viatura

dado o NIF de um utente, determina qual o montante global a pagar num determinado intervalo de datas, considerando todas as viaturas do utente

Figure 5.3: SUS interface-GesPark system

that the class `Utente` is defined in the sub-system `SubSistemaUtentes`. Be as complete as possible in constructing it, identifying the relationships, naming them and giving their multiplicities.

2. Add the **OCL** for the following constraint to the diagram in the previous paragraph, if necessary:

   All the records in a car park relate to vehicles registered in the system.

3. Construct the **Sequence Diagram** for the operation of the `GesPark` class which, given a identifier of a car and a date range, determines the amount to be paid in that date range.

   The operation must have the existence of the given identifier as a precondition. Add it to the class diagram.

4. Construct the **Sequence Diagram** for the operation of the `GesPark` class which, given an identifier, determines the total number of parks visited by the car with which it is associated.

   The operation should have the precondition that the identifier exists and the postcondition that the result is non-negative. Add them to the class diagram.

5. Now consider that the sub-system has information about the users who have registered in the car parks. Add this information to the architecture and build the **Sequence Diagram** of the operation that will allow you to calculate, for all the vehicles belonging to a user, identified by their NIF (VAT number), the total amount to be paid in a given date range.

   The operation must have the precondition that the user is registered in the sub-system. Add it to the Class Diagram.

Annex A

Code for Exercise 5.2.1:

```java
public class Shopping {
  private String name = "";
  private Map<String,Buyer> buyers; // idBuyer->Buyer
  ...
  public List<String> comprados(String bi) {
    List<String> res = null;
    boolean exists = this.buyers.containsKey(bi);
    if (exists)
        res = this.calculates(bi);
    return res;
  }
  public List<String> calculates(String bi) {
    Buyer c = this.buyers.get(bi);
    List<String> res = c.getBillets();
    return res;
  }
  ...
}

public class Buyer {
  private Map<String, List<String>> tickets; // Show -> Ticket
  ...
  public List<String> getBillets() {
    List<String> res = new ArrayList<String>();
    for (List<String> bs: this.tickets.values()) {
      res.addAll(bs);
    }
    return res;
  }
  ...
}
```

Appendix B

Code for Exercise 5.2.2:

```
public interface IGestTurmas {
   public Student getStudent(String codStudent)
   float getStudentMark(String codStudent);
   void registerDelivery(Delivery e, String codGroup);
   boolean validateEvaluators();
}

public interface Identifiable {
   String getID();
}

public abstract class Person {
   protected String name;
   public abstract void setName(String n);
}

public class Student extends Person implements Identifiable {
   private Group myGroup;
   private String numStudent;
   private int gradeTeo;
   private int bounsPrat;
   public void register(Group g) {...}
   public String getID() {...}
   public void setName(String n) {...}
}

public class Group implements Identifiable {
   private String cod;
   private int note;
   private List<Delivery> deliveries;
   public void addDelivery(Delivery e) {...}
   public String getID() {...}
}
```

```
public class Delivery {
   private Date data;
   private int teacher_grade;
   private Student evaluator;
   private int grade_evaluator;
   private String comments;
}

public class Teacher extends Person implements Identifiable{
   private String cod;
   public String getID() {...}
   public void setName(String n) {...}
}

public class SSGesTurmasFacade implements IGestTurmas {
   private Teacher responsible;
   private List<Docente> docentes_praticas;
   private Map<String,Student> students;
   private List<Group> groups;
   ...
}
```