Chapter 11

Exercise 11.1

Define a class **GeometricFigure** and three sub-classes **Square**, **Circle** and **Rectangle**. Define a method **Area** the implementation of which in **GeometricFigure** returns the value zero, while in the sub-classes it is evaluated as a function of the properties of the given sub-classes. Show the invocation of a method by a program that scans a list of geometric figures of an arbitrary nature.

Sol:

```
Add class GeometricFigure
     method Init() is public
     method Area(): float is public;

Add class Square inherits GeometricFigure
     type tuple (side : float)
     method Init(s: float) is public
     method Area(): float is public

Add class Rectangle inherits GeometricFigure
     type tuple (height: float
                 length: float)
     method Init(h: float, l:float) is public
     method Area(): float is public

Add class Circle inherits GeometricFigure
     type tuple (radius : float)
     method Init(r: float) is public
     method Area(): float is public

body Init(s:float) in class Square is public
     co2{ self->side=s }$

body Init(h:float, l:float) in class Rectangle is public
     co2{ self->height=h;
          slef->length=l; }$

body Init(r:float) in class Circle is public
     co2{ self->radius=s }$

body Area(): float in class GeometricFigure is public
     co2 {return 0}$

body Area(): float in class Square is public
     co2 {return ((self->side)*(self->side));}$

body Area(): float in class Rectangle is public
     co2 {return ((self->height)*(self->length));}$
```

```
body Area(): float in class Circle is public
     co2 {return ((self->side)*(self->side)*3.14);}$
```

If L is a list containing objects of class GeometricFigure (Circle, Square, Rectangle), we can write simply

```
     float a;
     for X in L do
          { a=X.Area();
            cout << a << endl; }
```

to display the values of the areas of each Figure in the list. The correct method will be applied to each object (late binding).

Exercise 11.2

Define a data-dictionary of an object-oriented database. Suggest the introduction of classes and hierarchies concerning various concepts of an object oriented schema, classes, atomic types, types structured using various constructors, generalisation hierarchies and methods with their input/output parameters. Populate the data dictionary with data that describe part of the schema dealing with the management of automobiles, described in Figure 11.2. Then think of a query that allows, for example, the extraction of a list of the classes and methods with covariant redefinition of the output parameter.

Sol:

This is an object-oriented database schema that represents the data dictionary of an OO-DBMS

```
add class Class
     type tuple (Name: string,
                 Parents: set(Class),
                 Properties: set (Attribute),
                 Methods: set (Method));

add class Attribute
     type tuple (Name: string,
                 Property: Type);

add class Type;

add class Atomic_Type inherits Type
     type tuple (Name: string);

add class Object_Type inherits Type
     type tuple (Class: string);

add class Tuple_Type inherits Type
     type tuple (set(Attribute));

add class List_Type inherits Type
     type tuple (element: Type,
                 n_of_element: integer);

add class Set_Type inherits Type
     type tuple (Element: Type);

add class Method
     type tuple (Name: string,
                 Class: Class,
                 input: list(Type),
                 output: Type);
```

With reference to figure 11.2, the OO-DBMS contains the following objects:

```
O1: <OID1, [“string”]> of type Atomic_Type
O2: <OID2, [“integer”]> of type Atomic_Type
O3: <OID3, [“RegistrationNumber”,OID1]> of type Attribute
O4: <OID4, [“Model”,OID1]> of type Attribute
O5: <OID5, [“Color”,OID1]> of type Attribute
O6: <OID6, [“Price”,OID2]> of type Attribute
O7: <OID7, [“Motor”,OID1]> of type Attribute
O8: <OID8, [“ShockAbsorber”,OID1]> of type Attribute
O9: <OID9, [“Name”,OID1]> of type Attribute
O10: <OID10, [“Name”,OID1]> of type Attribute
O11: <OID11, [“City”,OID1]> of type Attribute
O12: <OID12, [“NoOfEmployee”,OID2]> of type Attribute
O13: <OID13, [“ManufactureDate”,OID2]> of type Attribute
O14: <OID14, [“MaxSpeed”,OID2]> of type Attribute
O15: <OID15, [“Name”,OID1]> of type Attribute
O16: <OID16, [“Address”,OID1]> of type Attribute
O17: <OID17, [“TaxCode”,OID1]> of type Attribute
O18: <OID18, [{OID7,OID8}]> of type Tuple_Type
O19: <OID19, [OID1]> of type Set_Type     (* Past Victories*)
O20: <OID20, [“Mechanical Parts”,OID18]> of type Attribute
O21: <OID21, [“Person”,{}, {OID15,OID16,OID17},{}]> of type Class
O22: <OID22, [“Factory”,{}, {OID10,OID11,OID12},{}]> of type Class
O23: <OID23, [OID22]> of type Set_Type
O24: <OID24, [“OID21”]> of type Object_Type
O25: <OID25, [“Manufactures”,{},{OID9,OID23,OID24},{}]> of type
                                                              Class
O26: <OID26, [“OID25”]> of type Object_Type
O27: <OID27, [“Automobile”,{},{OID3, OID4, OID26, OID5, OID6,
                                         OID20 }, {}]> of type Class

O28: <OID28, [“VintageCar”,{OID27},{OID13},{}]> of type Class
O29: <OID29, [“SportCar”,{OID27},{OID14,OID24},{}]> of type Class
O30: <OID30, [“PastVictories”,OID19]> of type Attribute
O31: <OID31, [“VintageSportCar”,{OID28,OID29},{OID14},{}]> of type
                                                              Class
```

<u>Query:</u>

```
select X.Name
from X in Class, Y in Class, Z in Attribute
where X in Y.Parents
 and  Z in X.Properties
 and  Z in Y.Properties

select X.Name
from X in Methods, Y in Class, Z in Class
where Y in Z.Parents
 and X in Y.Methods
 and X in Z.Methods
```

Exercise 11.3

Consider the following schema of an **O2** object oriented database:

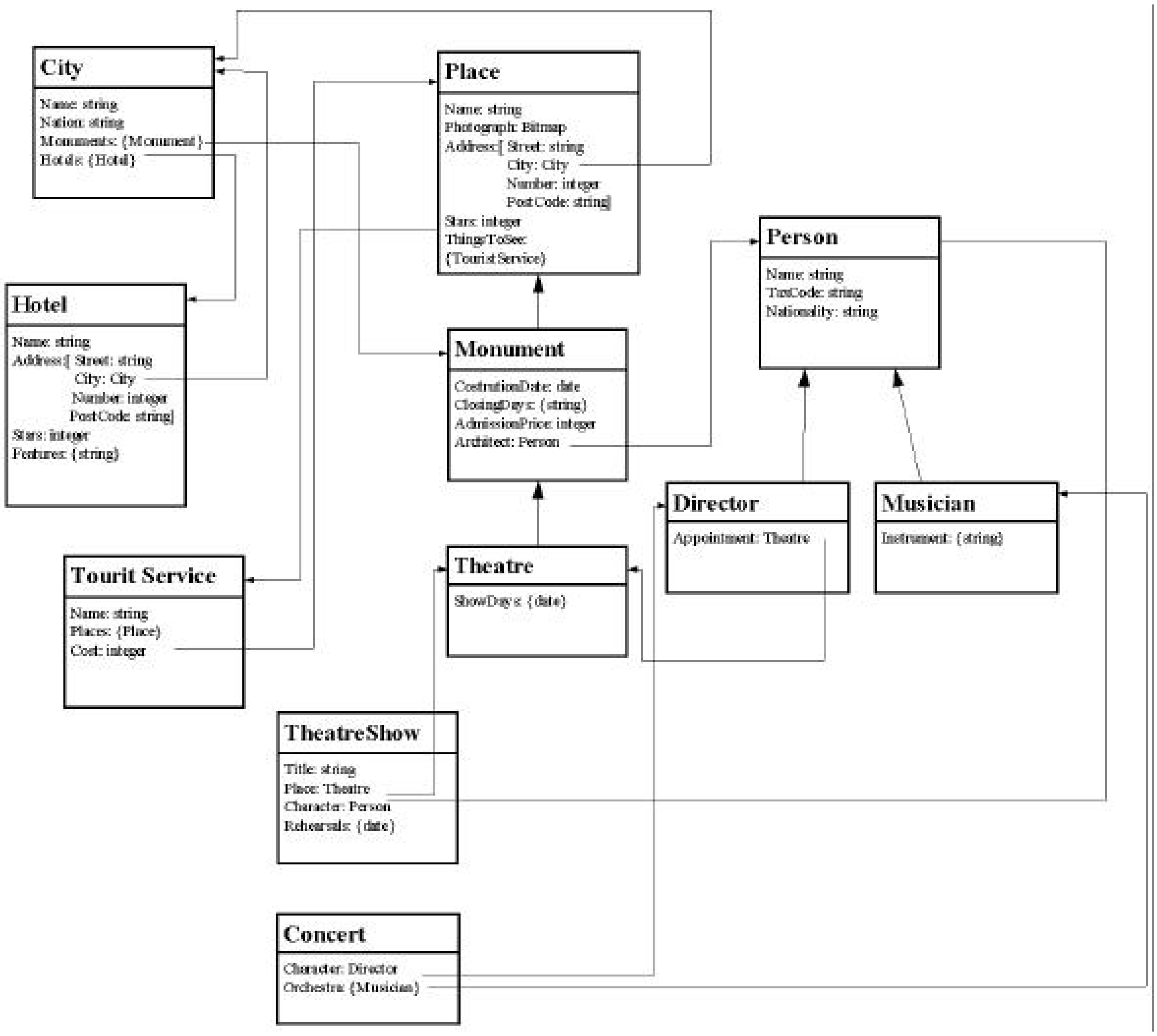
```
add class City
     type tuple (Name: string,
                 Nation: string,
                 Monuments: set(Monument),
                 Hotels: list(Hotel));

add class Hotel
     type tuple (Name: string,
                 Address: tuple (Street: string,
                                 City: City,
                                 Number: integer,
                                 PostCode: string);
                 Stars: integer,
                 Features: list(strings));

add class Place
     type tuple (Name: string,
                 Photograph: Bitmap,
                 Address: tuple (Street: string,
                                 City: City,
                                 Number: integer,
                                 PostCode: string);
                 ThingsTosee: set(TouristService));

add class Monument inherits Place
     type tuple (ConstructionDate: date,
                 ClosingDays: list(string),
                 AdmissionPrice: integer
                 Architect: Person);

add class TouristService
     type tuple(Name: string,
                Places: set(Place),
                Cost: integer);

add class Theatre inherits Monument
     type tuple(ShowDays: list(date));

add class TheatreShow
     type tuple(Title: string,
                Place: Theatre,
                Character: Person,
                Rehearsal: set(date));

add class Concert inherits TheatreShow
     type tuple(Character: Director,
                Orchestra: set(Musician));
```

```
add class Person
     type tuple(Name: string,
                TaxCode: string,
                Nationality: string));

add class Director inherits Person
     type tuple(Appointment: Theatre);

add class Musician inherits Person
     type tuple(Instruments: set(string));
```

1) Graphically describe the above schema as illustrated in figure 11.2
2) Define the initialization method s for the classes Place, Monument and Theatre, reusing the methods while descending along the generalisation hierarchy.
3) Which property of the schema is described in a covariant way ?
4) Define the signature of the initialization method of a theatre show and then refine the signature in a covariant way in the input parameters whenever the show is a concert.
5) Give an example of invocation of the method defined above, in which it is not possible to verify its accuracy at the time of compilation.

Sol:

1)



2)

```
method init(n:string, ph:Bitmap, st:string, c:City,
  num:integer,pc:string, pl:set(TouristService)) in class Place is
  public

body init(n:string, ph:Bitmap, st:string, c:City, num:integer,
  pc:string, pl:set(TouristService) in class Place is public
    co2{ self->Name=n;
         self->Photograph=ph;
         self->Address.Street=st;
         self->Address.City=c;
         self->Address.Number=num;
         self->Address.PostCode=pc;
         self->ThinsToSee=pl; }$

body init (n:string, ph:Bitmap, st:string, c:City, num:integer,
  pc:string, pl:set(TouristService), cd:Date, close:set(string),
  adm:integer, ar:Person) in class Monument is public
    co2{ self init@Place(n,ph,st,c,num,pc,pl);
         self->Constructiondate=cd;
         self->ClosingDays=close;
         self->AdmissionPrice=adm;
         self->Architect=ar; }$

body init (n:string, ph:Bitmap, st:string, c:City, num:integer,
  pc:string, pl:set(TouristService), cd:Date, close:set(string),
  adm:integer, ar:Person, d:list(date)) in class Theatre is public
    co2{ self
           init@Monument(n,ph,st,c,num,pc,pl,cd,close,adm,ar,d);
         self->ShowDays=d; }$
```

3) The field Character in Class TheatreShow is redefined in Class Concert.

4)

```
method init(T:string, Pl:Theatre, ch:Person, reh:set(date)) in
  class TheatreShow is public

method init  (T:string, Pl:Theatre, ch:Director, reh:set(date)) in
  class Concert is public
```

5) The two init methods in class TheatreShow and Concert have the same name and the same input parameters (the Director is a Person). So the init method of class Concert may be applied to an object that is a TheatreShow. It is not possible at compilation time to verify this situation, because the difference between Person and Director will be known only at execution time, when the methods will be applied with their effective parameters.

Exercise 11.4

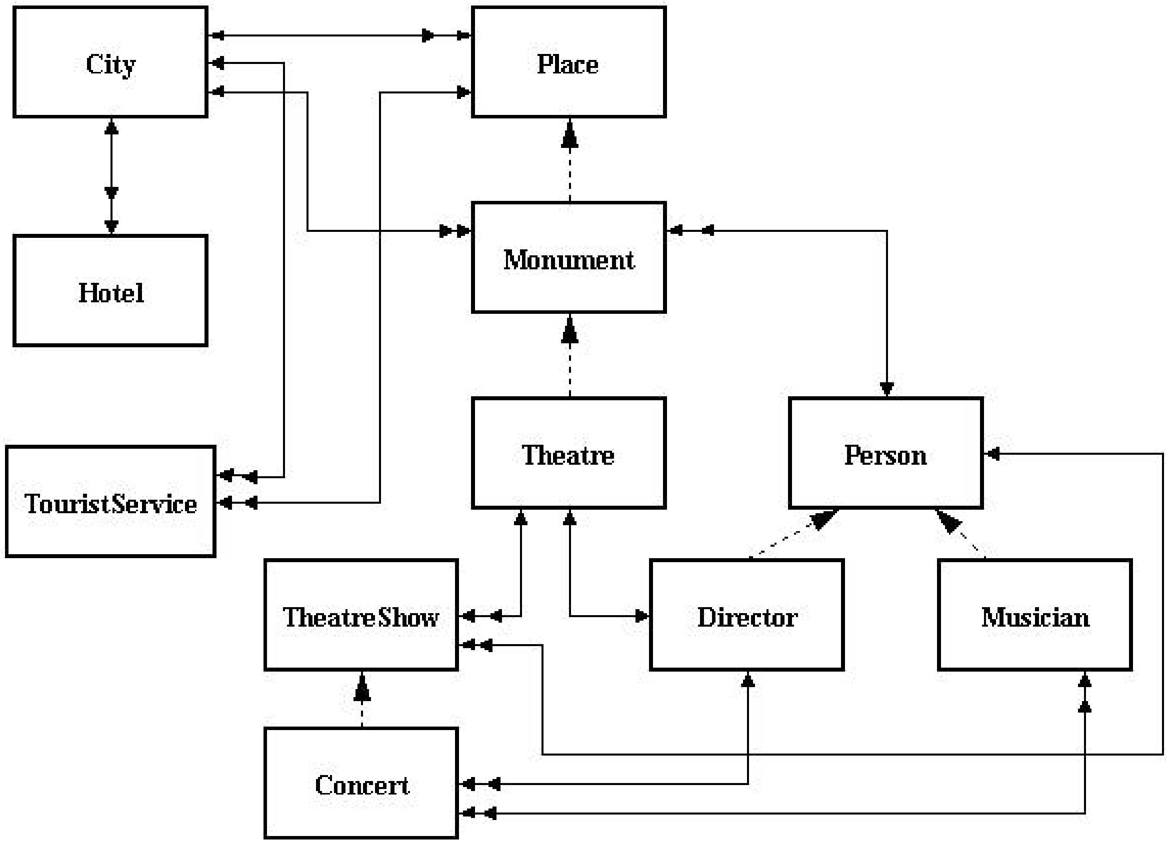
Describe the schema of the object-oriented database of Exercise 11.3 using the standard ODMG-93. Describe it graphically using the technique illustrated in figure 11.4

Sol:

```
interface City
     { attribute string Name;
       attribute string Nation;
       relationship set<Monument> Monuments
               inverse Monument::City;
       relationship list<Hotel> Hotels;
               inverse Hotel::Address.City
       relationship set<Place> Places;
               inverse::Place::Address.City }

interface Hotel
     {attribute string Name;
      attribute structure Address {
                    string Street;
                    relationship City City
                         inverse Hotels::City;
                    integer Number;
                    string PostCode; }
      attribute integer Stars;
      attribute list<string> Features; }

inteface Place
     { attribute string Name
       attribute Bitmap Photograph;
       attribute structure Address {
                    string Street;
                    relationship City City
                         inverse Hotels::City;
                    integer Number;
                    string PostCode; }
       relationship set<TouristService> ThingsToSee
          inverse Touristservice::Places;   }


interface Monument {extent Place}
      { attribute date ConstructionDate;
        attribute list<string> ClosingDays;
        attribute integer AdmissionPrice
        relationship Person Architect
           inverse Person::Monuments;
        relationship City City
           inverse City::Monuments  }
```

```
interface TouristService
      { attribute string name;
        relationship set<Place> Places
           inverse Place::TouristService;
        attribute integer Cost;   }

interface Theatre {extend Monument}
      { attribute set<date> ShowDays;
        relationship set<TheatreShow> Shows
          inverse TheatreShow::Place;
        relationship Director Director
          inverse Director::Appointment;  }

interface TheatreShow
      { attribute string Title;
        relationship Theatre Place
           inverse Theatre::Shows;
        relationship Person Character
            inverse Person::Characters;
        attribute set<date> Rehearsales; }

interface Concert {extent TheatreShow}
      { relationship Director::Character
           inverse Director::Concerts;
        relationship set<Musician> Orchestra
           inverse Musician::Concerts }

interface Person
      { attribute string Name;
        attribute string TaxCode;
        attribute string Nationality;
        relationship set<Monument> Monuments
           inverse Monument::Architect;
        relationship set<TheatreShow> Characters
           inverse TheatreShow::Character; }

interface Director {extent Person}
     {  relationship Theatre Appointment
           inverse Theatre::Director;
        relationship set<Concert> Concerts
           inverse Concert::Character; }

interface Musician {extent Person}
     { attribute set<string> Instruments;
       relationship set<Concert> Concerts
           inverse Concert::Orchestra; }
```

Interface Schema:



Exercise 11.5

With reference to the object-oriented database schema in Exercise 11.3, write the following queries in OQL:

1) Extract the names of the four stars hotels in Como.
2) Extract the names and the costs of tourist services offered in Paris.
3) Extract the names of the five stars hotels in the cities in which concerts conducted by Muti are planned.
4) Extract the names of the monuments in Paris created by Italian architects.
5) Extract the tourist services on offer partly in Paris and partly in another city.
6) Extract the names of the artistic directors of a theatre where no concerts are presented.
7) Extract the title of the concert, conductor, musicians and instruments used by each musician in the concerts of 12-2-99 in Milan.
8) Extract the cities having more than 10 monuments and fewer than 5 hotels.
9) Extract the name of the French architects who are also musicians.
10) Extract the total number of concerts conducted by Muti in either Italian or French theatres.
11) Extract the total number of concerts given in each Italian theatre.
12) Classify the monuments in Paris according to the date of construction. Use the classification: "Renaissance" (from 1450 to 1550), "Baroque" (from 1550 to 1800), "Imperial" (from 1800 to 1900), "Modern" (from 1900 to today), and count the number of element of each class.

Sol:

```
1)  select distinct x.name
    from x in hotels
    where x.Address.City.name="Como"
    and x.Stars=4

2)  select distinct x.name, c.cost
    from x in TouristService, y in Place
    where y in x.Places
    and y.Address.City.Name="Paris"

3)  select distinct x.Name
    from x in Hotel
    where x.Stars=5
    and x.Address.City in ( select y.Place.Address.City
                            from y in concert
                            where y.Character.Name="Muti")

5)  select distinct x.Name
    from x in TouristService, y in Places
    where "Paris" in x.Places.Address.City.Name
    and y.Address.City.Name<>"Paris"
    and y in x.Places
```

```
6)  select distinct x.Name
    from x in Director
    where x.Appointment not in
          ( select y.Place
            from y in Concert )

7)  select distinct struct
          { Title: x.Title
            Conductor: x.Character
            Musicians: ( select distinct struct
                           { Name: y.Name
                             Instruments: y.Instruments
                            }
                         from y in Musicians
                         where y in x.Musicians )
          }
     from x in Concert
     where “12-2-99” in x.Rehearsales
     and x.Place.Address.City.Name=”Milan”

8)   select distinct x
     from x in City
     where 10  < ( select count
                  from y in Monument
                  where y.Address.City=x )
     and 5 >   ( select count
                 from z in Hotel
                 where z.Address.City=x )

9)   select distinct x.Name
     from x in Musicians
     where x.Nationality=”Frech”
     and x in ( select distinct y.Architect
                from y in Monument )

10)  select count
     from x in Concert
     where x.Character.Name=”Muti”
     and (x.Place.Address.City.Nation=”Italy” or
          x.Place.Address.City.Nation=”France” )

11)  group x in
          ( select y
            from y in Concert
            where y.Place.Address.City.Nation=”Italy” )
     by (Theatre: x.Place)
     with (Number: count ( select z
                           from z in Partition ))
```

```
12)  group x in
          ( select y
            from y in Monument
            where y.Address.City.Nation=”Paris”)
     by (Reinassance: 1450 < x.ConstructionDate.year < 1550,
         Baroque:   1550 <= x.ConstructionDate.year < 1800,
         Imperial:  1800 <= x.ConstructionDate.year < 1900,
         Modern:    x.ConstructionDate.year >=1900 )
     with (Number: count ( select z
                           from z in partition ) )
```

Exercise 11.6

Use the SQL-3 syntax to describe the object model presented in Exercise 11.3 (represent the O2 list as sets).

Sol:

```
create row type CityType (
     Name varchar(30) primary key,
     Nation varchar(30),
     Monuments setof(ref (MonumentType)),
     Hotel setof(HotelType)
);

create table City of type CityType;

create row type AddressType (
     Street varchar(30),
     City ref(CityType),
     Number integer,
     PostCode varchar(30)
);

create row type HotelType (
     Name varchar(30),
     Address AddresType,
     Stars integer,
     Features setof(varchar(40)),
     primary key (Name,Address)
);

create table Hotel of type HotelType;

create row type PlaceType (
     Name varchar(30),
     Address AddresType,
     ThingsToSee setof (ref(TouristServiceType)),
          primary key (Name,Address)
);

create table Place of type PlaceType;

create row type TouristServiceType (
     Name varchar(30) primary key,
     Places setof(ref(PlaceType)),
     Cost integer
);

create table TouristService of type TouristServiceType;
```

```
create row type MonumentType (
     ConstructionDate date,
     ClosingDays setof(varchar(30)),
     AdmissionPrice integer,
     Architect ref(PersonType)
) under PlaceType;

create table Monument of type MonumentType under Place;


create row type TheatreType (
     ShowDays setof(date)
) under MonumentType;

create table Theatre of type TheatreType under Monument;


create row type TheatreShowType (
     Title varchar(30) primary key,
     Place ref(TheatreType),
     Character ref(PersonType),
     Rehearsales setof(date)
);

create table TheatreShow of type TheatreShowType;


create row type ConcertType (
     Character ref(Director),
     Orchestra setof (MusicianType)
) under TheatreShowType;

create table Concert of type ConcertType under TheatreShow;

create row type PersonType (
     Name varchar(30),
     TaxCode varchar(30) primary key,
     Nationality varchar(30)
);

create table Person of type PersonType;

create row type DirectorType (
     Appointment ref(TheatreType)
) under PersonType

create table Director of type DirectorType under Person;
```

```
create row type MusicianType (
     Instruments setof(varchar(30))
) under PersonType;

create table Musician of Type MusicianType under Person;
```

Exercise 11.7

Considering the SQL-3 database schema introduced in the previous exercise, express the following queries in SQL-3.

1) Retrieve the names of the cities having “Liechtenstein” as nation.
2) Retrieve the names of the musicians playing in the concerts directed by Karajan.
3) Retrieve the names of the monuments in London constructed in the 17th Century and closed on Monday.
4) Retrieve the names of the directors who perform at theatres different from those to which they are affiliated.
5) Retrieve, from each theatre, the title of all the concerts that are planned for the year 2000.

Sol:

1)
```
select Name
from City
where Nation=“Lienchtenstein”
```

2)
```
select Name
from Musician, Concert
where Concert->Character.Name=“Karajan“
and Name in Concert.Orchestra.Name
```

3)
```
select Name
from Monument
where Address..City.Name=”London”
and 1601 <= ConstructionDate <= 1700
and “Monday” in ClosingDays
```

4)
```
select Character
from TheatreShow
where Character in Director
and Character->Appointement <> Place
```

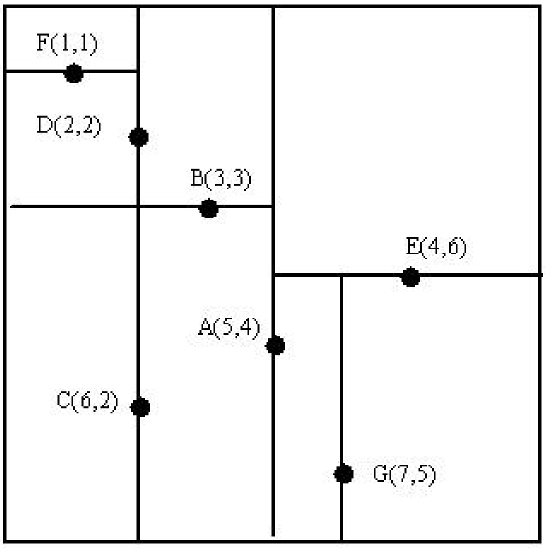
5)
```
select Title, Place
from TheatreShow
where 2000 in Rehearsals.year
```

Exercise 11.8

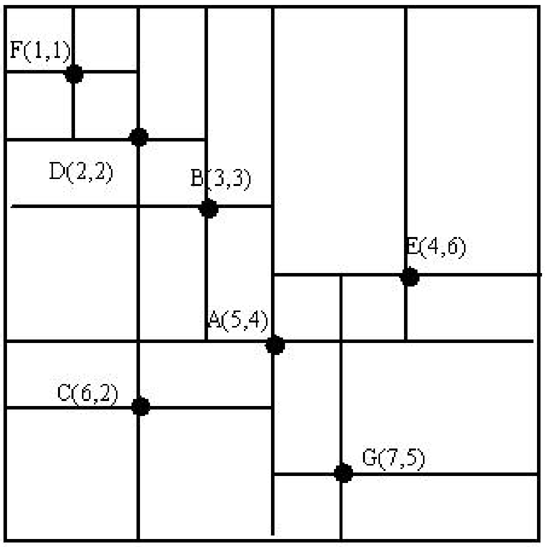
Build a 2d-tree and quad-tree representation of the sequence of bidimensional points: A(5,4), B(3,3), C(6,2), D(2,2), E(4,6), F(1,1), G(7,5).
How many intermediate nodes appear, in the two representation, between A and F and between A and G ?

Sol:

2d-tree



quadtree



In the 2d-tree representation there are two nodes between A and F (D and B) and 1 node between A and G (E).

Also in the quad-tree representation there are 2 nodes between A and F (B and D), but there are no nodes between A and G.

Exercise 11.9

With reference to the object-oriented database schema of Exercise 11.3, indicate a choice of complex indexes for the efficient management of the path expressions that are most used by the queries of Exercise 11.5

Sol:

The path expressions most used by the queries are:

```
Place.Address.City.Name,
Place.Address.City.Nation,
Hotel.Address.City.Name.
```

All the queries refers to the last values of the paths, not to the intermediate values.
A nested index to these path may be a good solution to make the queries more efficient.