Object-Oriented Database Design using UML and ODMG

Object Oriented Databases (I) – Lecture 9

Advanced Databases

Lecture outline

- Database Design Process
- Object-Oriented concepts
  - Objects, Classes
  - Attributes, Operations, Associations
  - Encapsulation, Inheritance
- Object-Oriented Data Modelling
  - Identifying
    - Classes, attributes, and operations.
    - Associations among classes
  - Drawing class diagrams – conceptual model
- Introduction to ODMG 3.0 (the standard for Object-Oriented Databases)

References

Database Systems – 4th Edition (chapters 25 to 27) by Connolly & Begg, Addison Wesley, 2005

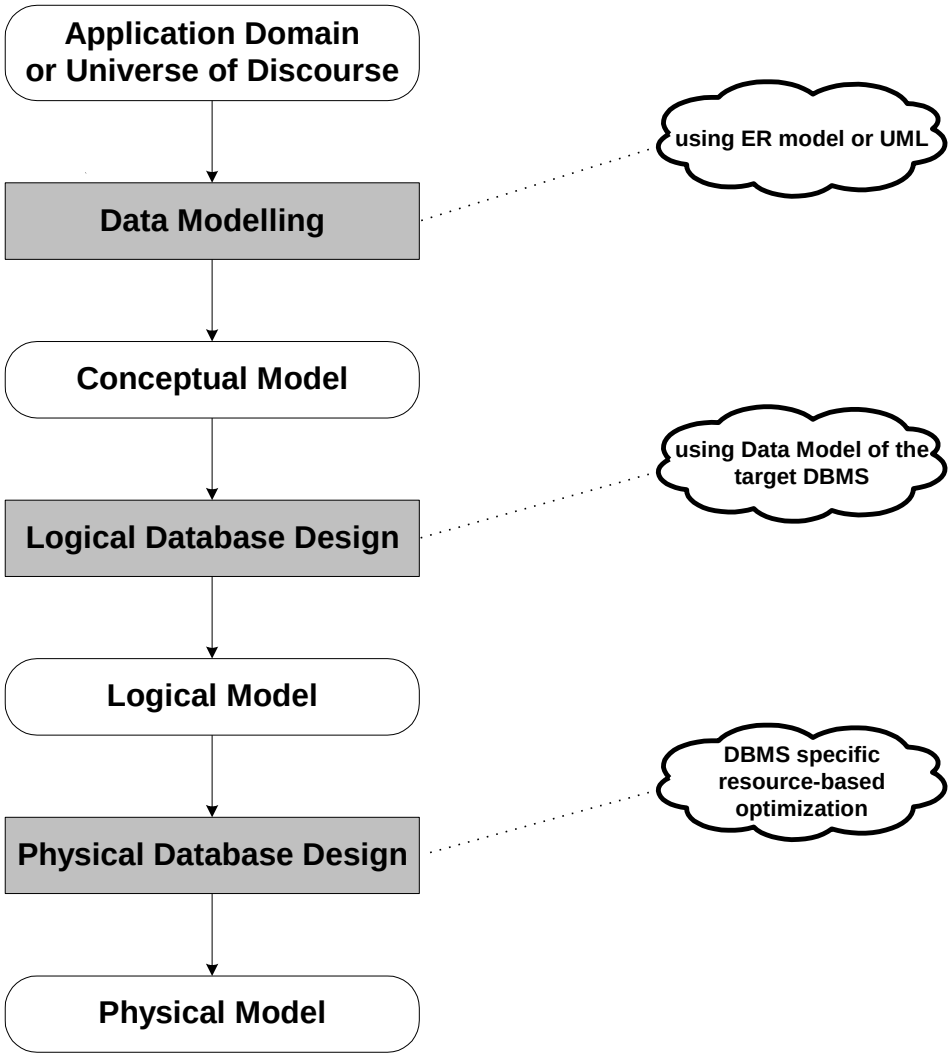
Database Design for Smarties using UML for Data Modelling (chapter 13) by Robert J. Muller, Morgan Kaufmann Publishers, 1999 (specialist text in library)

Fundamental of Database Systems – 5th Edition (chapters 20 & 21) by R. Elmasri and S. B. Navati, Addison Wesley, 2007

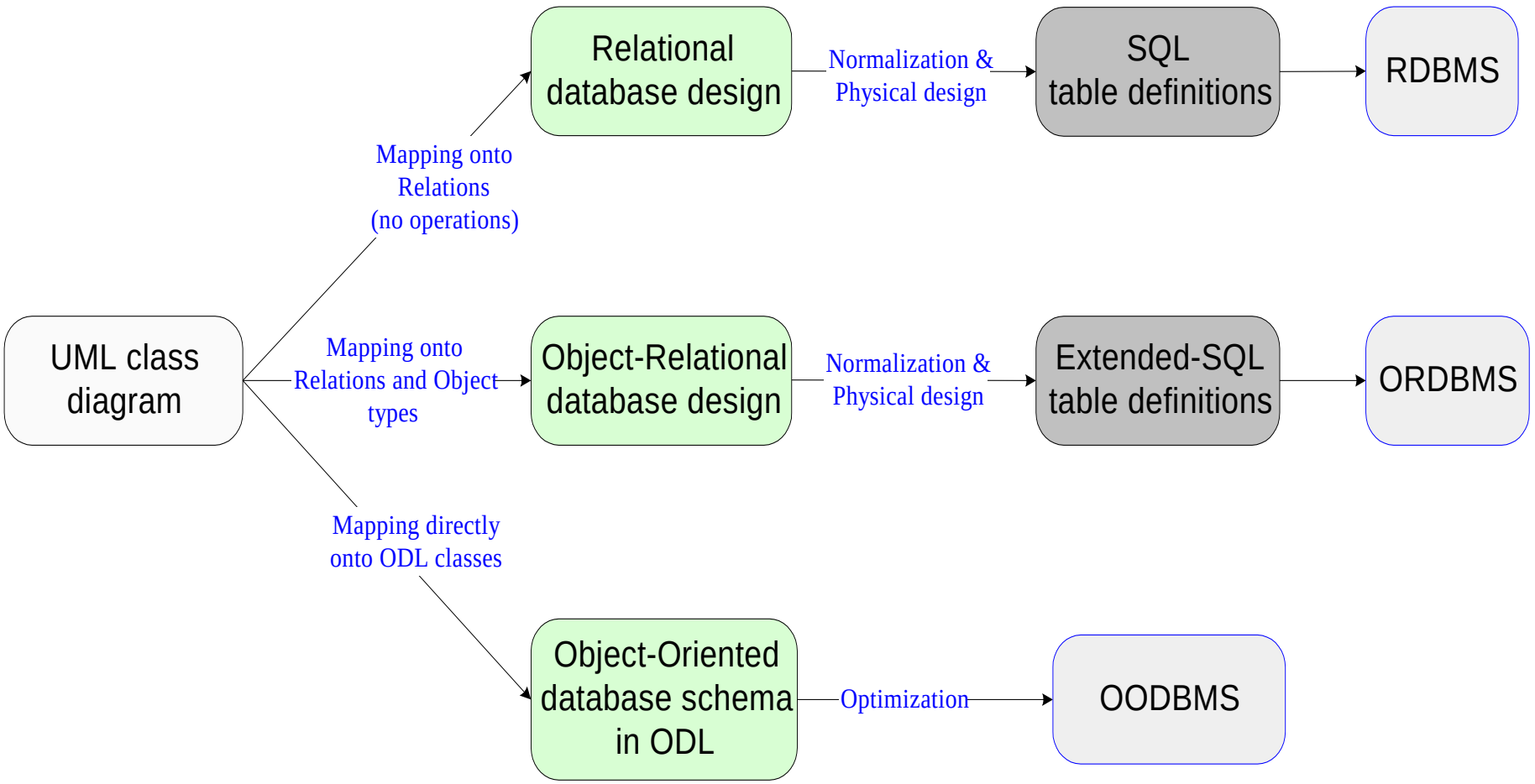
Object Database Standard: ODMG 3.0 by R.G.G. Cattell, Douglas K. Barry, Morgan Kaufmann Publishers, 2000 (reference in library)

Object-Oriented Database Design clearly explained by Jan L. Harrington. Morgan Kaufmann Publishers, 2000 (reference in library)

Database Design Process



Logical/Physical database design



Object-oriented concepts

Objects

- Objects represent real world entities, concepts, and tangible as well as intangible things.
  - For example a person, a drama, a licence
- Every object has a unique identifier (OID).
  - System generated
  - Never changes in the lifetime of the object
- An object is made of two things:
  - **State:** attributes (name, address, birthDate of a person)
  - **Behaviour:** operations (age of a person is computed from birthDate and current date)
- Objects are categorized by their type or class.
- An object is an instance of a type or class.

Object-oriented concepts …

Classification

Classification is the process of grouping together objects which have common features.

Programming languages have type systems and database systems have data models to classify object.

The name used for the classificatory group of values is usually called ***class***.

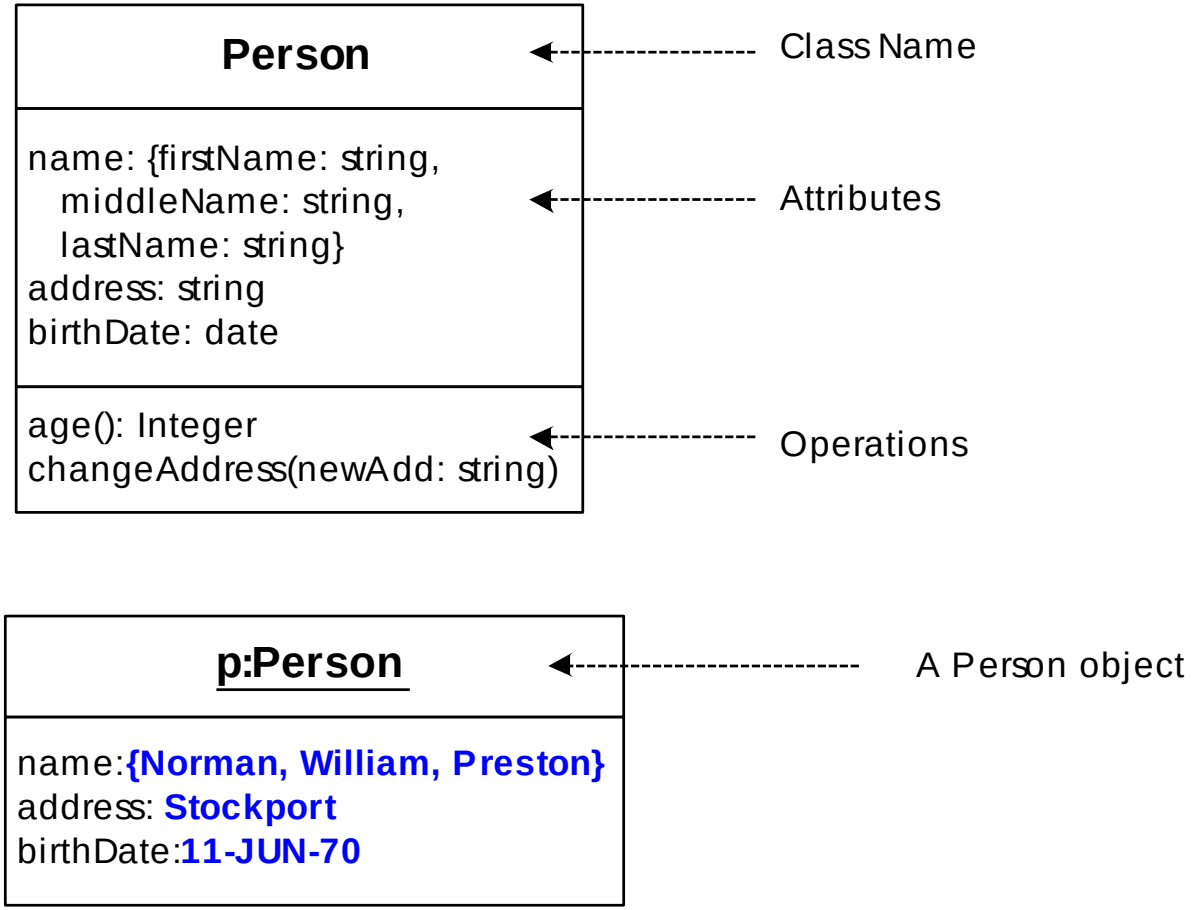
Class

Provides a template for constructing objects.

Instances of a class have the same kind of data and identical behaviour.

Object-oriented concepts ...

An Example of a class in UML



Object-oriented concepts ...

Encapsulation

Merger of data structure and operations.

Objects are composed of attributes (values) and operations (behaviour).

Inheritance

A class can be defined

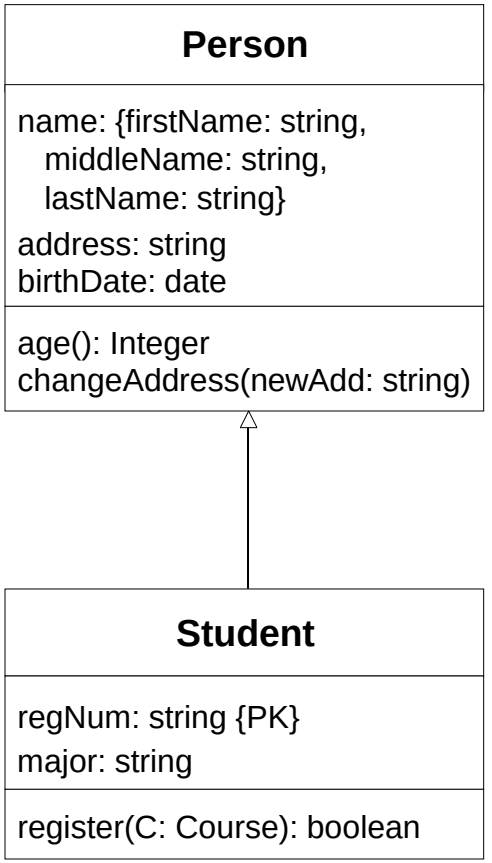
in terms of another one.

Person is super-class

and Student is sub-class.

Student class inherits attributes

and operations of Person.



Object-oriented concepts ...

An *object system* or *object-based system* is one that supports the modeling of data as abstract entities, with object identity.

An *object-oriented system* is an object system in which all data is created as instances of classes which take part in an inheritance hierarchy.

An *object-oriented database management system* (ODBMS) is a DBMS with an object-oriented logical data model.

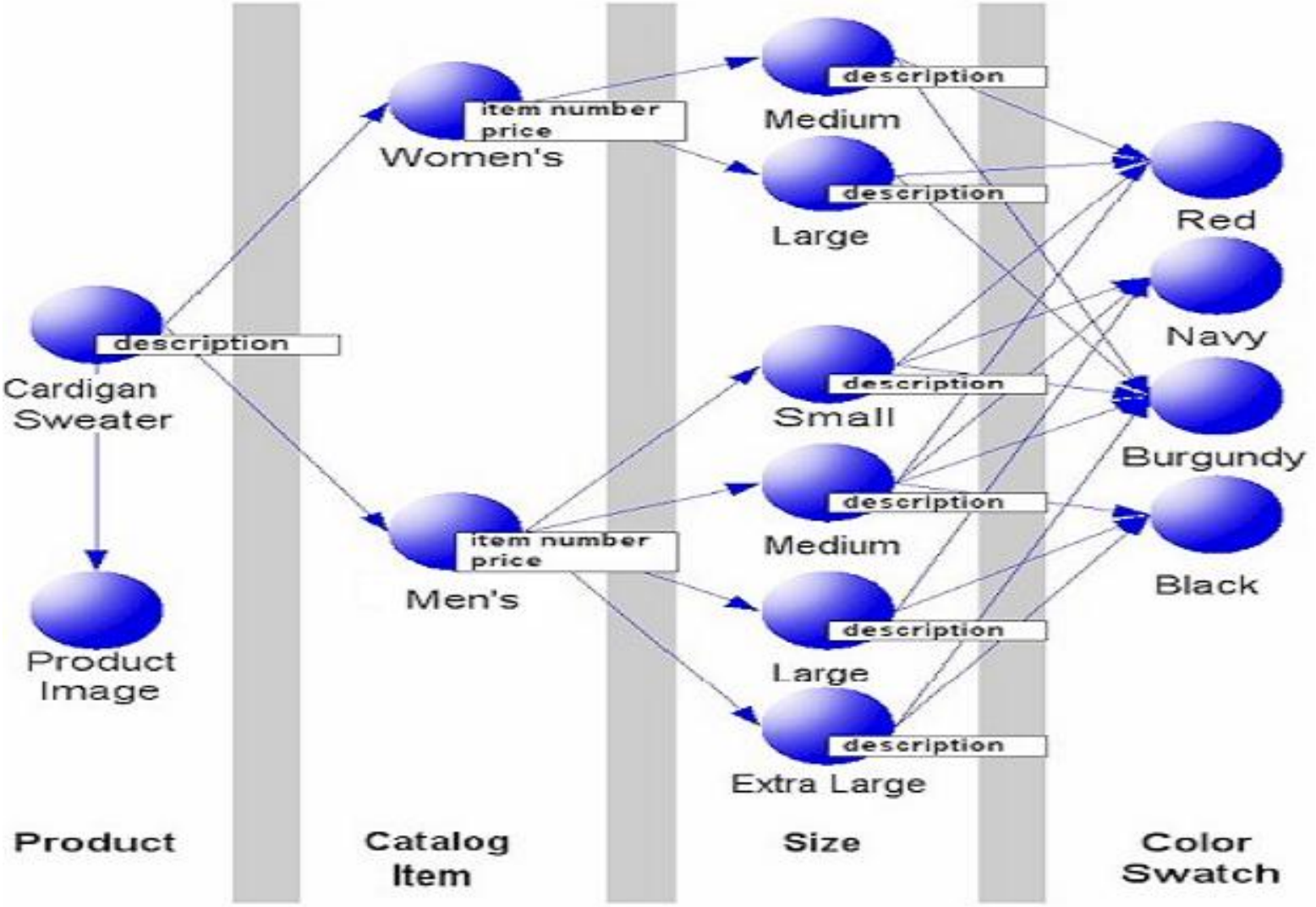
An *object-oriented database* (ODB) is a database made up of objects and managed by an ODBMS.

Why ODBs?

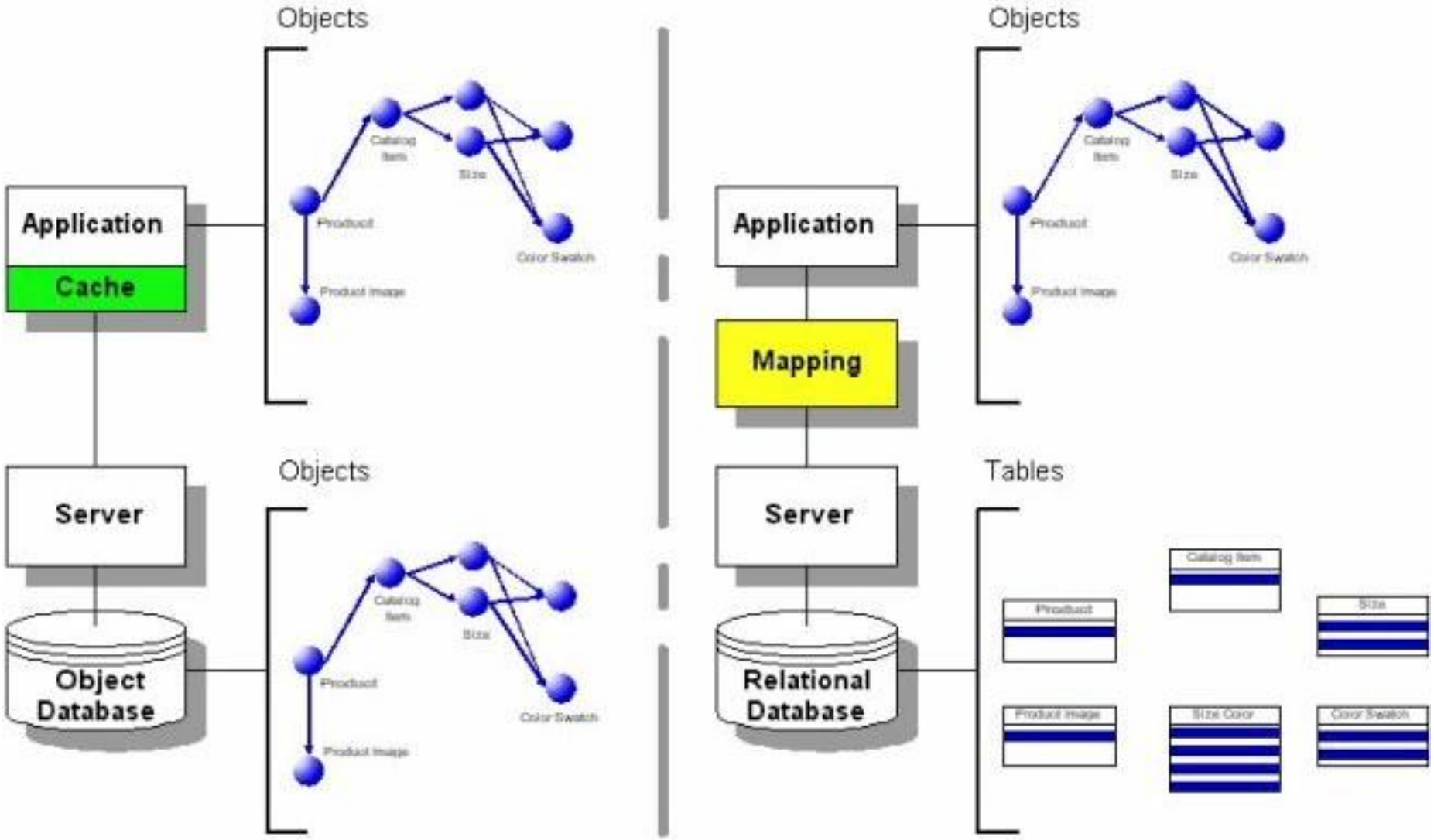
ODBs are inevitable when:

- Data is complex and variable in size
- Complex structural and compositional relationships
- Data is highly inter-related
- Data is evolving rapidly over time
- Richer data types
  - complex objects
  - inheritance
  - user extensibility
- Behaviour with data
  - not just a *data* model but also
  - operations can be bundled together with data

Complex Data



ODBs are more Natural & Direct



Comparison

RDBs vs. ORDBs

- Very easy to compare because both are based on Relational Model.
- An RDB does not support abstract data types (ADT), all attribute values must be atomic and relations must be in first normal form (flat relation).
- An ORDB supports ADTs, attributes can be multi-valued, and does not require first normal form.
- The underlying basic data structures of RDBs are much simpler but less versatile than ORDBs.
- ORDBs support complex data whereas RDBs don’t.
- ORDBs support wide range of applications.

Comparison – continued...

RDBs vs. ODBs.

- Not very easy to compare because of philosophical differences.
- RDBs have only one construct i.e. Relation, whereas the type system of ODBs is much richer and complex.
- RDBs require primary keys and foreign keys for implementing relationships, ODBs simply don't.
- ODBs support complex data whereas RDBs don't.
- ODBs support wide range of applications.
- ODBs are much faster than RDBs but are less mature to handle large volumes of data.
- There is more acceptance and domination of RDBs in the market than that for ODBs.

Comparison – continued...

ODBs vs. ORDBs.

- Both support ADTs, collections, OIDs, and inheritance, though philosophically quite different.
- ORDBs extended RDBs whereas ODBs add persistence and database capabilities to OO languages.
- Both support query languages for manipulating collections and nested and complex data.
- SQL3 is inspired from OO concepts and is converging towards OQL (object query language).
- ORDBs carries all the benefits of RDBs, whereas ODBs are less benefited from the technology of RDBs.
- ODBs are seamlessly integrated with OOPLs with less mismatch in the type systems;
- ORDBs (SQL3) have quite different constructs than those of OOPLs when used in embedded form.

Object-Oriented Data Modelling

- Identification of objects in the system
  - Use UML analysis techniques e.g. use-cases, domain object models.
  - Potential sources are:
    - Things, People, Roles, Organizations, Concepts
    - Events, Processes, Places, Locations, etc
  - Devise an Object Model
- Refining the object model
  - Grouping objects in Classes
  - Identifying Attributes, Operations, Associations & Multiplicities
  - Drawing class diagrams (of **persistent** classes)
- Reconciling classes
  - Revisiting the classes for inheritance
  - Considering normalization of classes into simple classes
  - Producing a big picture: a class diagram (perhaps without showing attributes and operations).

OO Data Modelling: Example

Universe of Discourse: **TU Information System (TU-IS)**

Tribhuwan University has several academic departments.

Each department provides one or more courses.

Each course is composed of several modules, where a module may be part of more than one course.

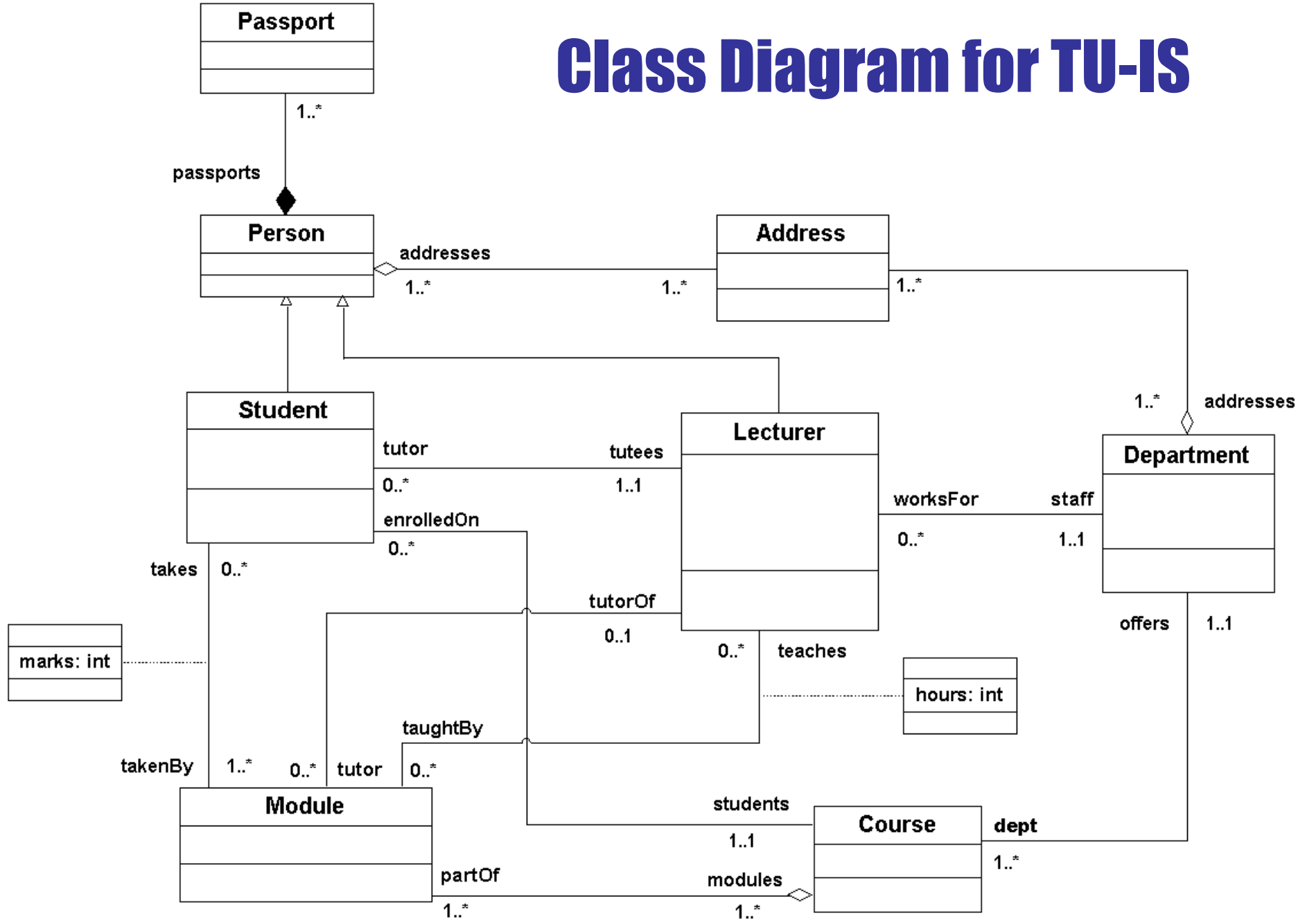
A student enrolls on a course and every year takes a specified number of modules. Note that several students are usually registered for a course.

Every student is assigned a tutor at the start of the course, who is a lecturer in the department providing the course.

A lecturer works for a department and usually teaches on several modules.

Each module has a module tutor who is a lecturer. A lecturer may be a tutor of several modules.

Class Diagram for TU-IS



ODMG 3 Object Database Standard

- Object Database Management Group, formed 1991, intended to:
  - provide a standard where previously there was none
  - support portability between products
  - standardize model, querying and programming issues
- Enables both designs and implementations to be ported between compliant systems
- Currently on version 3.0
- Most vendor products are moving toward compliance; O2 is possibly the closest
- Vendors
  - Object Design,Objectivity, O2 Technology, POET, etc.
- URL: www.odmg.org
- We will be using lambda-DB, a freely available ODBMS.

ODMG Components

An architecture for OODBMS.

An object model.

- that will act as the logical model for all OODBMS and provide a level of interoperability.

A data definition language (ODL).

- a concrete specification of the operations permitted over a schema defined in the data model.

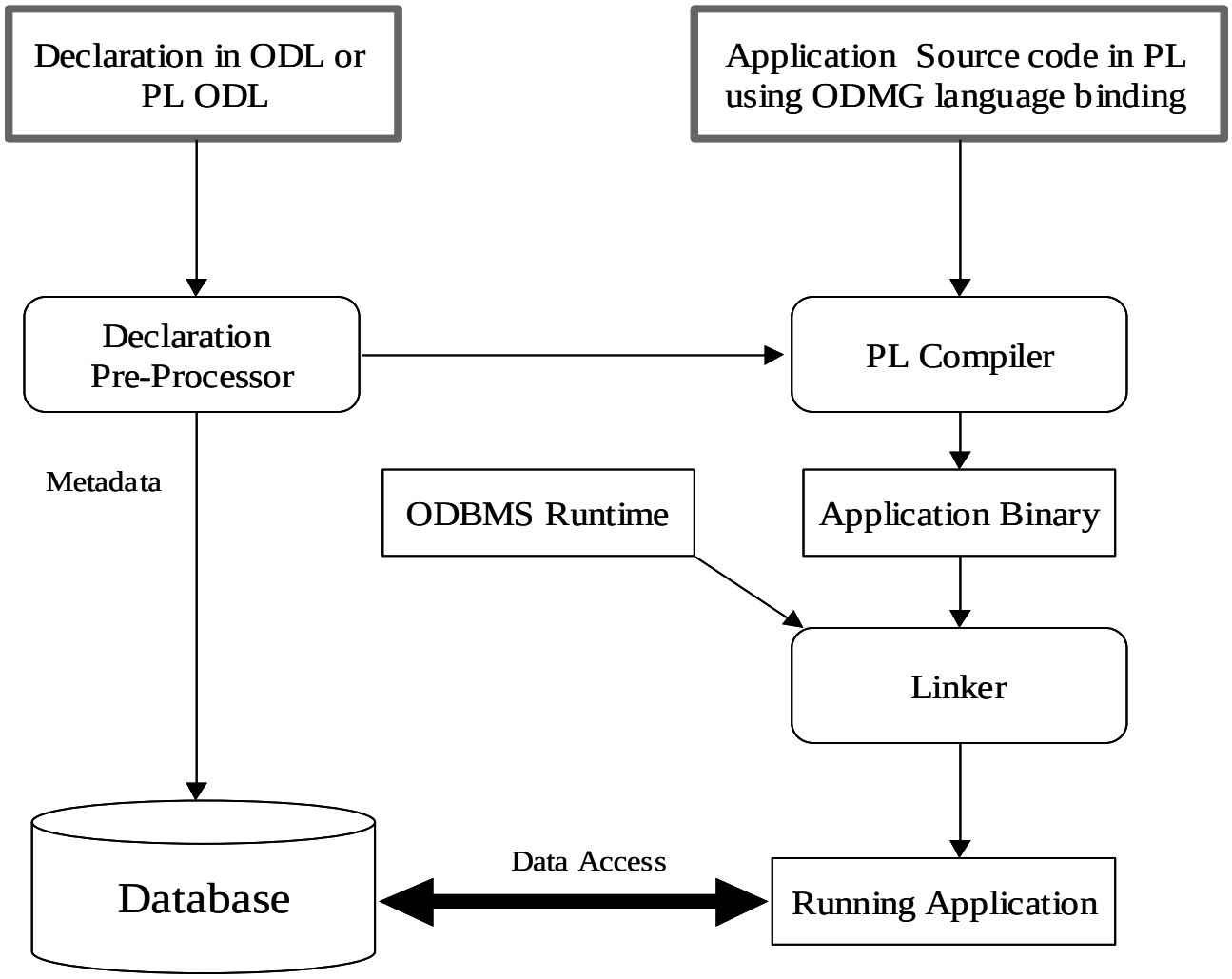
A query language (OQL).

- for posing ad-hoc queries but not for data definition or data manipulation.

Language bindings to existing OOPL (C++, Java, Smalltalk).

- the object manipulation languages are the integration of a PL with the ODMG model, so the OOPL's get persistence and the OODB gets a flexible and standard DB programming language.

An Architecture for OODBMS



Object Model

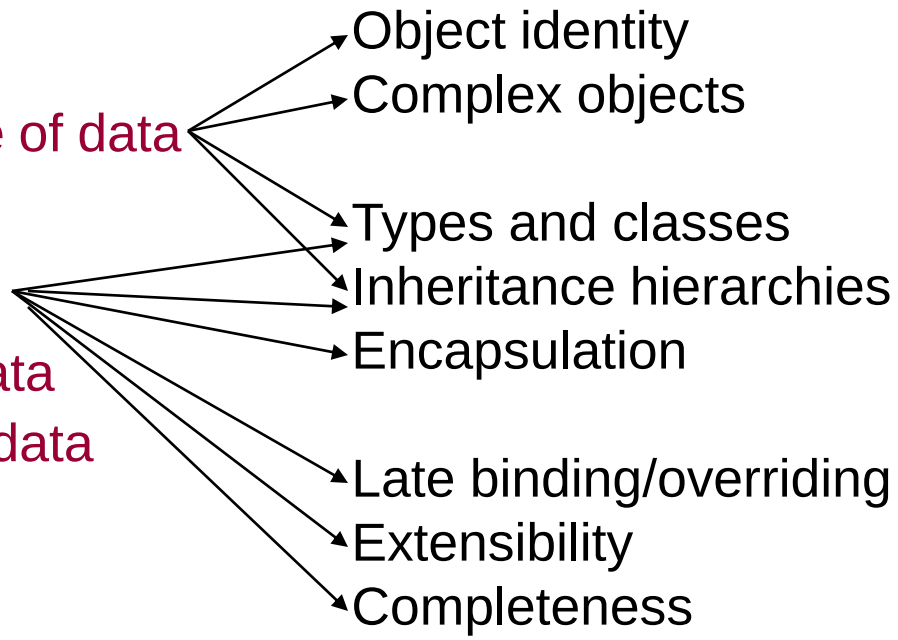
1. Data Model

   state and structure of data

2. Behaviour model

   dynamics of the data

   operations on the data



3. Persistence model

   the way the persistent and transient data is created & changes status

4. Naming model

   naming and accessing objects

Object Definition Language (ODL)

ODL is a specification language used to define the schema of an ODMG compliant database.

ODL supports all semantic constructs of the ODMG object model.

ODL is independent of any programming language, and hence provides means for the portability of database schema across complaint ODBMSs.

The database schema may comprise of:

- an ODL module (i.e. a higher level construct for grouping ODL specifications),
- some generic object types using interface,
- some concrete object types using class, and
- some literal types using struct, etc.

Components of ODL (literal)

Literal Types

Define values (not having OIDs)

Cannot stand alone i.e., must be embedded in objects

Can be simple, collection and structured

Simple

`long`, `short`, `unsigned long`, `unsigned short`, `float`, `double`, `char`, `string`, `boolean`, `enum`

Collection

`set`: unordered that do not allow duplicates,

`bag`: unordered that allow duplicates,

`list`: ordered that allow duplicates,

`array`: one-dimensional with variable length, and

`dictionary`: unordered sequence of key-and-value pairs without duplicate keys

Components of ODL (literal) …

Structured

`date`, `time`, `timestamp`, `interval`, and `struct`

For example

```
struct Name {
  string firstName,
  string middleName,
  string lastName
};
```

Components of ODL (object)

Object Types

`interface`: defines only the abstract behaviour of an object type.

Instances of an interface type cannot be created

For example

```
interface Object {
  ...
  boolean same_as(in Object other_object);
  Object copy();
  void delete();
};
```

`class`: defines both abstract state and behaviour of an object type

Instances of a class can be created

For example

```
class Person {
  ...
  attribute Name name;
  attribute date birthDate;
  unsigned short age();
};
```

Components of ODL (object) …

State definition: Attributes

An **`attribute`** is defined for each attribute in a UML class or an ER entity type.

An attribute belongs to a single class and is not a self-standing object.

The type of the values (domain) of an attribute is either object or literal (atomic, structured or collection).

For example:

```
attribute set<string> qualifications;
```

Defines an attribute of `Lecturer` class called `qualifications` the value of which is of type **`set<string>`**.

Consider that `lec` represents a `Lecturer` object then

```
lec.qualifications := set("BSc", "MSc", "PhD");
```

Will assign the set of strings as a value to the `qualifications` attribute of the lecturer object.

Components of ODL (object) ...

Behaviour definition: Operations

Objects may have certain behaviour that is specified as a set of operations.

An object type includes an **operation signature** for each operation that specifies:

- name of the operation,
- names and types of each argument, and
- the type of the returned value, if any.

For example:

```
unsigned short age();
```

Defines the operation `age` without any arguments which return a value of type **`unsigned short`**.

Components of ODL (object) …

Extent and Keys

Besides, attributes and operations, a class definition may specify an extent and a unique key.

Extent

Defines the set of all instances of a given class within an ODB.

Deleting an object removes the object from the extent of a corresponding class.

Key

Uniquely identifies the instances of a class.

The key concept is similar to the concept of primary key in RDBs, however, keys are not must in ODBs and are not used to implement relationships (as in the case of RDBs).

A class must have an extent to have a key.

For example:

```
class Student (extent Students key regNum) {...};
```

Defines `Students` to be the extent and `regNum` to be a unique key of the `Student` class.

Components of ODL (object) …

Atomic object type

Any user-defined object type e.g., `Person`

Collection object types

`Set`: unordered that do not allow duplicates,

`Bag`: unordered that allow duplicates,

`List`: ordered that allow duplicates,

`Array`: one-dimensional with variable length, and

`Dictionary`: unordered sequence of key-and-value pairs without duplicate keys

Structured object types

`Date, Time, Timestamp, Interval`

Watch out that ODL is case-sensitive e.g.,

`Set` is a collection object type whereas **`set`** is a literal collection.

`Name` is a type name whereas `name` is an attribute in the `Person` class definition.

Mapping Class Diagrams into ODL

At this stage, we are dealing with classes, attributes, and operations.

- Different associations and inheritance will be covered next.

Mapping (general case)

- Each UML class becomes an ODL class.
- Each attribute or method in a UML class becomes an attribute or operation of an ODL class with appropriate types.
- Specify a suitable extent name unless the class diagram explicitly indicates otherwise.
- Specify a unique key if one or more attributes of a UML class are shown in bold or tagged with {PK}.
- For a composite attribute, specify a structure literal type.

Mapping TU-IS class diagram into ODL

```
module TU_IS1 {
  struct Name {
    string firstName;
    string middleName;
    string lastName; };
  class Person {
    attribute Name name;
    attribute date birthDate;
    attribute char gender;
    unsigned  short age();  };

  class Lecturer (extent  Lecturers key lecturerId) {
    attribute string       lecturerId;
    attribute unsigned     short room;
    attribute float        salary;
    attribute date         joinedOn;
    attribute set<string>  qualifications;
    boolean   teachModule(in Module M); };
```

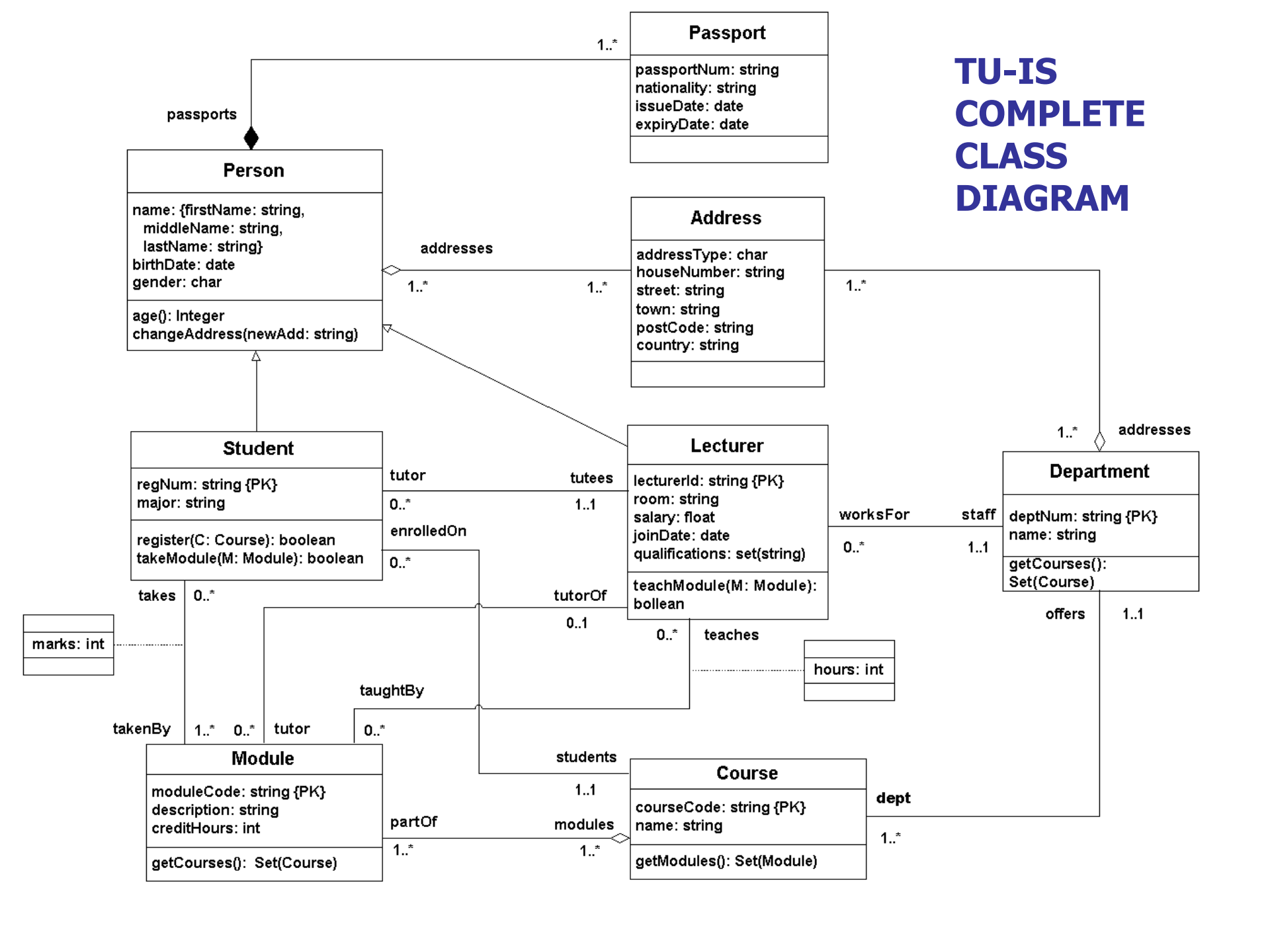
TU-IS schema in ODL …

```
class Department (extent Departments key deptNum) {
  attribute string deptNum;
  attribute string name; };

class Course (extent Courses key courseCode) {
  attribute string courseCode;
  attribute string name; };

class Module (extent Modules key moduleCode) {
  attribute string moduleCode;
  attribute string name;
  attribute unsigned short creditHours; };

class Student (extent Students key regNum) {
  attribute string regNum;
  attribute string major;
  boolean   register(in Course C);
  boolean   takeModule(in Module M); };
};
```


TU-IS COMPLETE CLASS DIAGRAM
Passport
passportNum: string
nationality: string
issueDate: date
expiryDate: date
1..*
passports
Person
name: {firstName: string, middleName: string, lastName: string}
birthDate: date
gender: char
age(): Integer
changeAddress(newAdd: string)
addresses
1..*
1..*
Address
addressType: char
houseNumber: string
street: string
town: string
postCode: string
country: string
1..*
Student
regNum: string {PK}
major: string
register(C: Course): boolean
takeModule(M: Module): boolean
tutor
tutees
0..*
1..1
enrolledOn
0..*
takes
0..*
Lecturer
lecturerId: string {PK}
room: string
salary: float
joinDate: date
qualifications: set(string)
teachModule(M: Module): bollean
worksFor
staff
0..*
1..1
1..*
addresses
Department
deptNum: string {PK}
name: string
getCourses(): Set(Course)
offers
1..1
tutorOf
0..1
0..*
teaches
marks: int
hours: int
taughtBy
takenBy
1..*
0..*
tutor
0..*
Module
moduleCode: string {PK}
description: string
creditHours: int
getCourses(): Set(Course)
students
1..1
partOf
1..*
modules
1..*
Course
courseCode: string {PK}
name: string
getModules(): Set(Module)
dept
1..*