Chapter 4

Data-Oriented Models

Chapter 4
Data-Oriented Models

- 4.1. Data Models: The ERD
- 4.2. Entity-Relationship Modeling
- 4.3. Object-Oriented Models
- 4.4. An Introduction to Objects
- 4.5. Object-Oriented v.s. Object-Based
- 4.6. Conceptual, Logical & Physical Models
- 4.7. Models as Communication Tools

4.1. Data Models: The ERD

- Mid-1970s: Databases were just coming into use
- Research to find ways to model data
- To give template for database design
- Many notations were tried
- Chen (1976) proposed the

Entity-Relationship Diagram (ERD)

The ERD was the first systems analysis tool to focus on

DATA

and

How it is Linked and Organized.

It was from this that objects then evolved.

4.1. Data Models: The ERD

- Chen made two contributions:
  - The Entity-Relationship **Principle,** and
  - The Entity-Relationship **Notation.**
- These further issues were also investigated by Chen and others:
  - E-R models for database design
  - The **significance** of the data of an enterprise
  - Data as a **corporate asset.**
  - The **stability** of the data of an enterprise

4.2. Entity-Relationship Modeling

The following six points all relate to both E-R and Object modeling. Now let’s examine them in this order:

❶ The Entity-Relationship **Principle.**

❷ E-R models for database design.

❸ The **stability** of data.

❹ The **significance** of data.

❺ Data as a ***corporate asset.***

❻ Entity-Relationship **Notation.**

➊ The Entity-Relationship *Principle*.

- The E-R principle focuses on the things we need to keep data **about.**
- Earlier methods emphasized **what the users need to know** to do their job.
- Here we emphasize what **things** the users need to know ***about.***
- The word **Entity** means a **thing.**
- *Later* we worry about what items they need to know ***about*** each entity.

➊ The Entity-Relationship *Principle*.

A **Data Entity** is some*thing* that has separate and distinct existence *in the world of the users* and is of interest to the users in that they need to keep data *about* it in order to do their job.

➊ The Entity-Relationship *Principle.*

We define:

An Entity Type is a class or category expressing the common properties that allow a number of entities to be treated similarly.

➊ The Entity-Relationship ***Principle*.**

- **Entity types are always singular.**
- **So our entity lists look like this:**

Sales

- Customer
- Product
- Sales clerk

Phone

- Customer
- Number
- Line
- Call
- Service

Booking

- Venue
- Artist
- Agent
- Concert
- Performance
- Customer
- Seat

All Singular

➊ The Entity-Relationship *Principle*.

An individual ***Customer*** or ***Product*** or ***Sale*** or ***Call*** or ***Artist*** is then an

Occurrence

of the Entity Type.

➊ The Entity-Relationship *Principle*.

- ***Attributes*** **are the data elements carried by an entity that describe it and record its state.**
- ***Attributes*** **are the things we need to know *about* an Entity.**

❶ The Entity-Relationship Principle.

Associations:

We define:

An **Association** is the interaction of two entities and is represented by a verb.

➊ The Entity-Relationship Principle.

Associations :

- The purpose is to provide access paths to the data.
- When a sale occurs, we will link:

A Customer

To a Product

To a Sales Clerk.

- Using a ***verb*** to describe each link.

➊ The Entity-Relationship Principle.

Associations:

In this way we are able to diagram the **structure** of our user's business data,

independent of any way that the data may be used,

either now or in the future.

❶ The Entity-Relationship Principle SUMMARY

- An **Entity** is a thing the users need to know (i.e, record) something about.
- An **Entity Type** is a group or class of entities that are all the same kind of thing.
- An **Occurrence** of an Entity Type is a specific individual thing of the kind the Entity Type describes.
- **Attributes** are the things we need to know about an Entity.
- An **Association** is the interaction of two Entities and is represented by a verb.
- These interactions among the Entities (i.e., these *associations*) show us the **pathways** we need to follow through the database to access the data.

❷ E-R models for database design.

There is a very significant reason why Entities, Attributes and Associations are so fundamentally important for systems development.

② E-R models for database design.

For database design:

- **Each *Entity* becomes a file or table.**
- **Each *Attribute* becomes a field (i.e., a column)**
- **Each *Association* becomes an access pathway (i.e., a foreign key)**

❸ The Stability of Data.

This tells us that

Data Entities Are Stable Over Time.

❸ The Stability of Data.

- However, entities are neither static nor stagnant;
- There may be an occasional new entity,
- There will often be new attributes,
- But as long as we stay in the

same business

there will be very

few new entities.

❸ The Stability of Data.

This stability contributes greatly to reducing the costs and delays in system maintenance.

❸ The Stability of Data.

However, ***beware new business!***

Mergers, acquisitions, takeovers and diversifications can all put your company into

new businesses

with a host of

new entity types.

④ The Significance of Data.

The Data modeler's Creed:

Data

is the

Centre of the Universe

➄ Data as a Corporate Asset.

- Since data is something the users *must have* in order to work,
- It can rightly be regarded as a **Business Asset** or **Resource**
- Just like
  - Finance
  - Human Resources
  - Plant and Equipment
  - Vehicles
  - Inventories
  - Etc.

➎ Data as a ***Corporate Asset.***

Like any corporate asset, there are some basic functions that must be done as part of managing the asset:

- Acquisition
- Organizing, storing and safekeeping
- Deployment, on time, to the people who need it
- Disposal when no longer needed

⑤ Data as a *Corporate Asset.*

- This has led to two new functions in the information industry,
    - Data Administration (DA) and
    - Data Resource Management (DRM).
- DA is a middle-management function concerned with finding and documenting every data element the company uses.
- DRM is a high-level management function, involving long-term strategic planning. DRM reports to the CIO (Chief Information Officer; in a well-organized company the CIO is a vice-president).

❺ Data as a *Corporate Asset.*

- The focus of DA and DRM is to manage data as we would manage any other business resource.
- The ERD has been the tool used to understand, document and administer data.
- As of now, the Object Model is taking over this function.

⑥ The Entity-Relationship Notation.

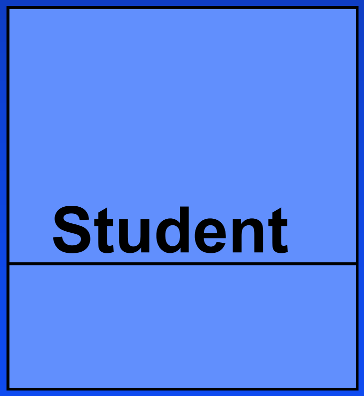
- There are many E-R notations.
- All of them work, any will do the job.

- Use whichever one you or your boss or your professor or your client prefers.

- UML is now the standard notation in the object world, so
- We will use UML for our Entities as well as our Objects.

❻ The Entity-Relationship Notation. Entities

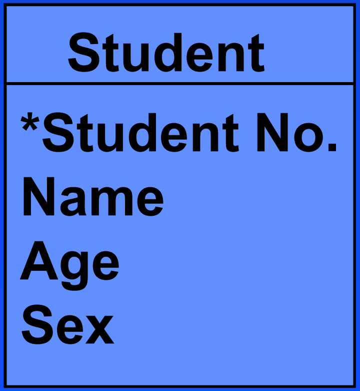
- Draw an Entity as a square or rectangular box, divided by a line near the top.
- Above the line place the name, singular.



⑥ The Entity-Relationship Notation. Attributes

- If there are not too many, Attributes may go in the bottom part of the box.
- Primary key first, marked with an asterisk.
- If not enough room, show only the key,

  list the others in the accompanying write-up.



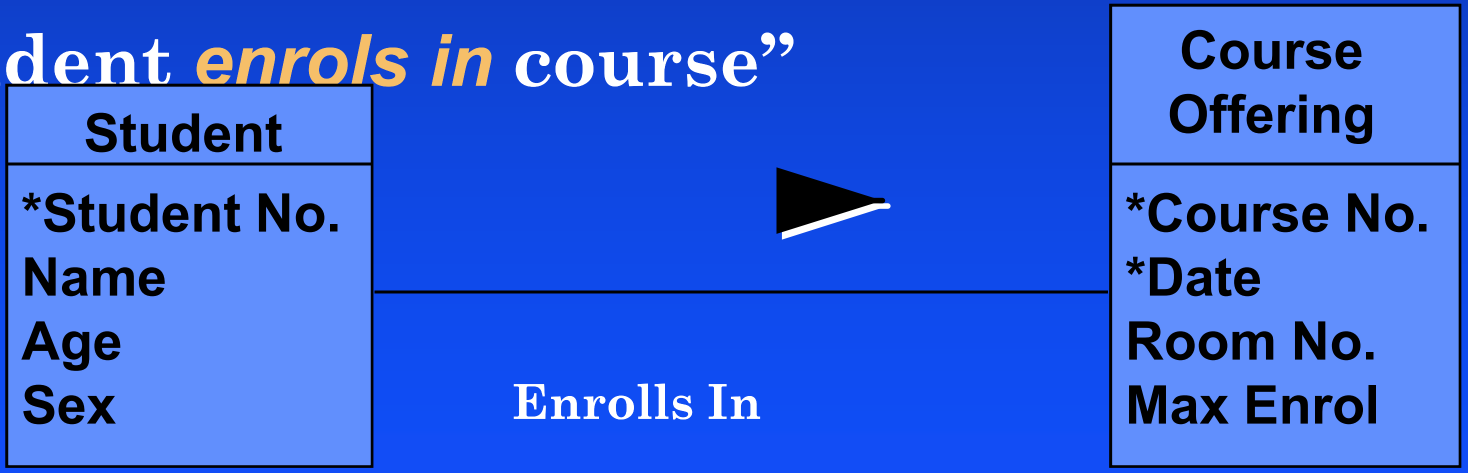
Student

*Student No.
Name
Age
Sex

⑥ The Entity-Relationship Notation.

Associations:

- Draw a line joining two entity boxes to show a relationship exists
- Write the verb above the line.
- Add a solid arrowhead so that it makes a sentence when you read it **in the direction of the arrow**:
- "Student *enrols in* course"



Student

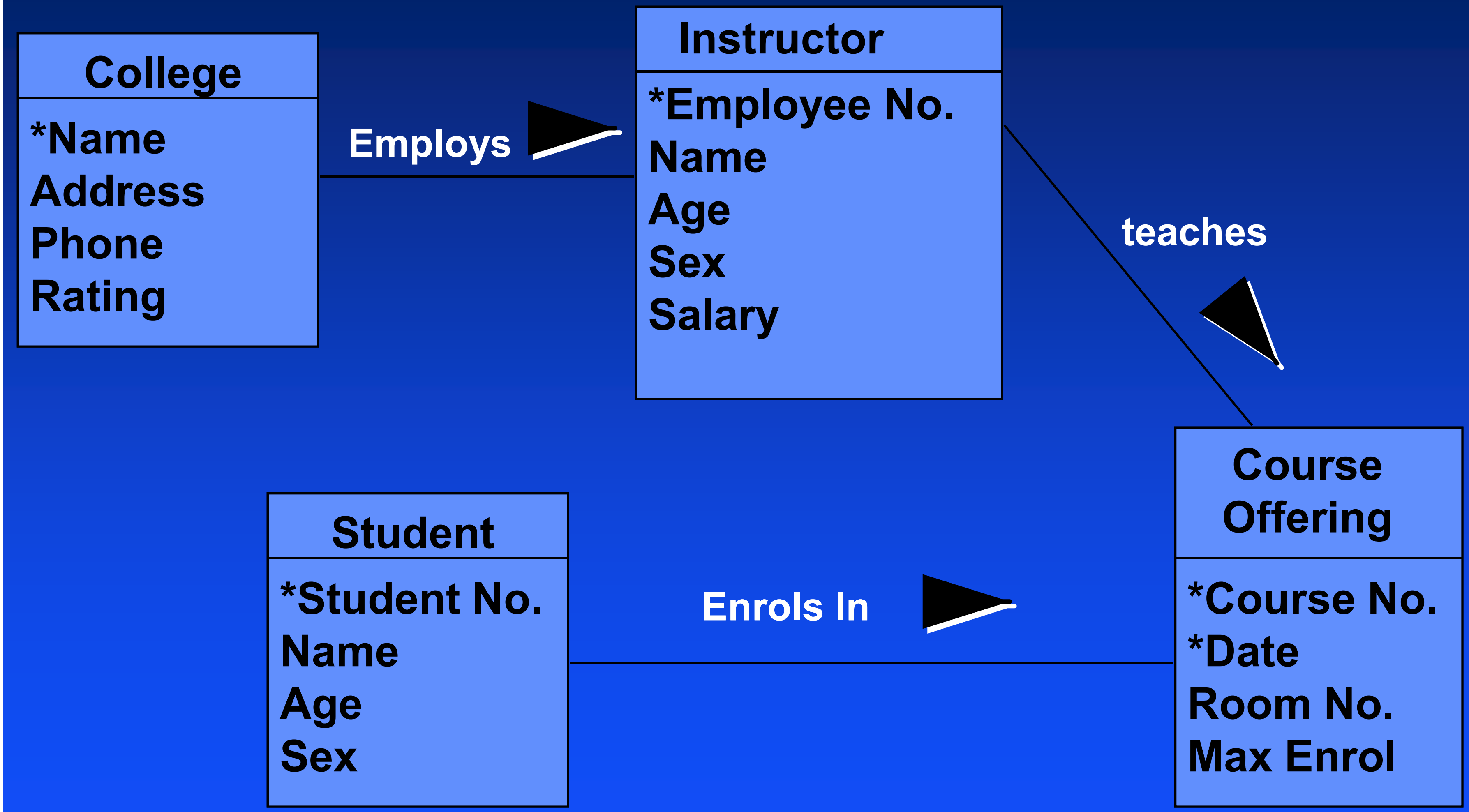
*Student No.
Name
Age
Sex

Course Offering

*Course No.
*Date
Room No.
Max Enrol

⑥ The Entity-Relationship Notation.

Associations



College

*Name
Address
Phone
Rating

Employs

Instructor

*Employee No.
Name
Age
Sex
Salary

teaches

Course Offering

*Course No.
*Date
Room No.
Max Enrol

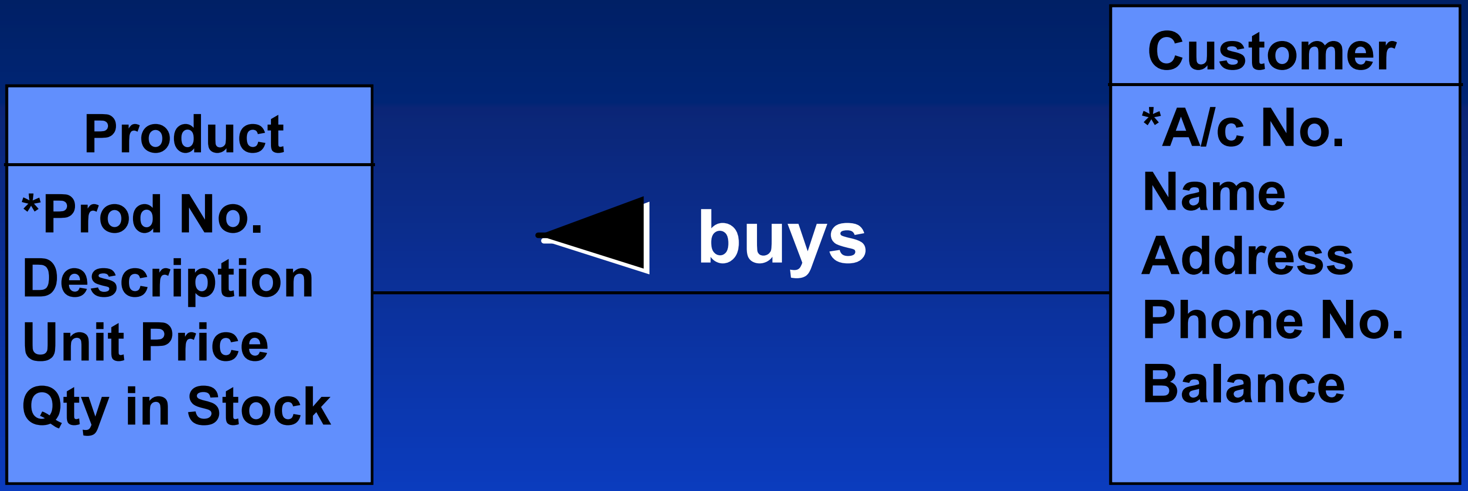
Student

*Student No.
Name
Age
Sex

Enrols In

⑥ The Entity-Relationship Notation.

More Examples of Associations



Product

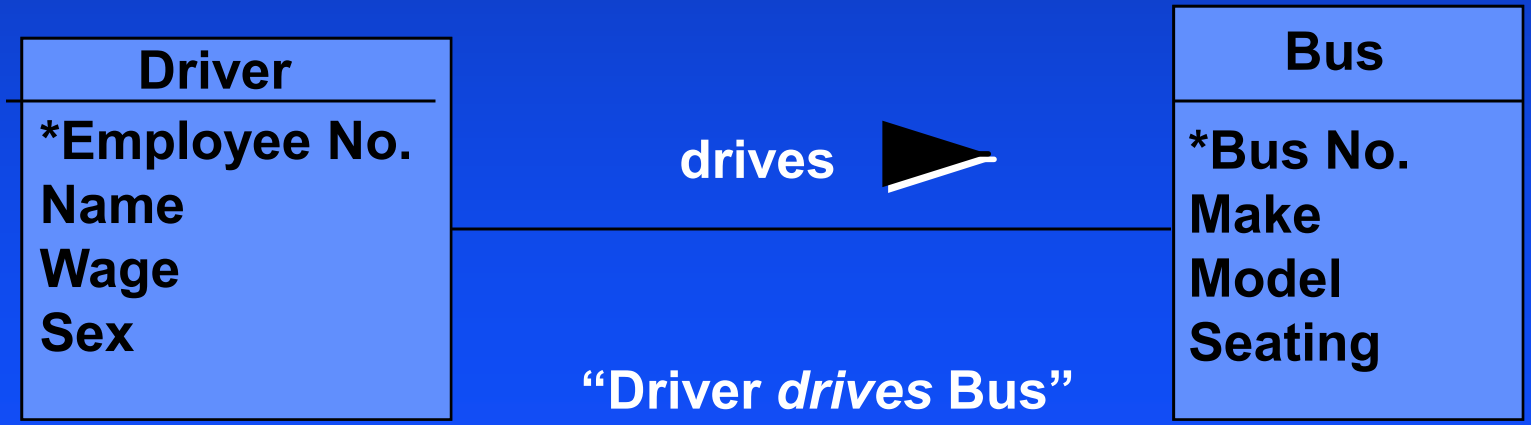
*Prod No.
Description
Unit Price
Qty in Stock

◀ buys

Customer

*A/c No.
Name
Address
Phone No.
Balance

“Customer *buys* Product”



Driver

*Employee No.
Name
Wage
Sex

drives ▶

Bus

*Bus No.
Make
Model
Seating

“Driver *drives* Bus”

Read the sentence in the direction of the arrow.

⑥ The Entity-Relationship Notation. Multiplicity

- The critical thing we need to know is

Is it One, or

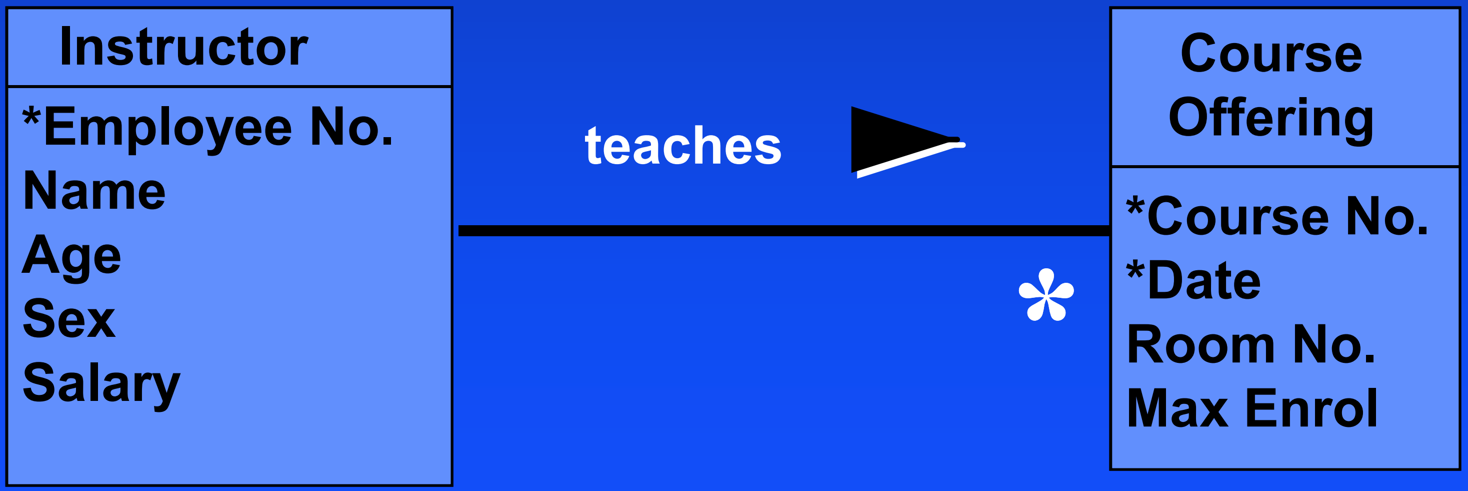
Is it Many?

- This is the ***Multiplicity*** of the relationship (also called *cardinality*).

❻ The Entity-Relationship Notation. Multiplicity

- We diagram this by adding a star (asterisk) below the relationship line whenever it should show “many.”
- Read this one as

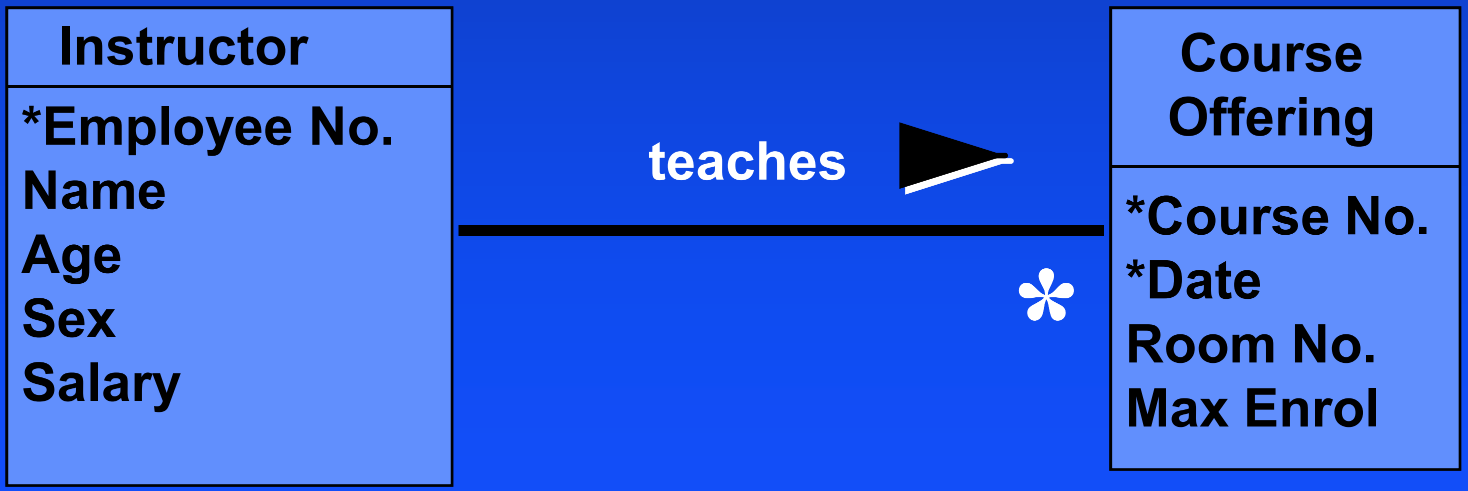
“Instructor *teaches* **many** course offerings”



⑥ The Entity-Relationship Notation. Multiplicity

- We refer to this as a

“One-to-Many” Association, or 1:M



Instructor

*Employee No.
Name
Age
Sex
Salary

teaches ▶

*

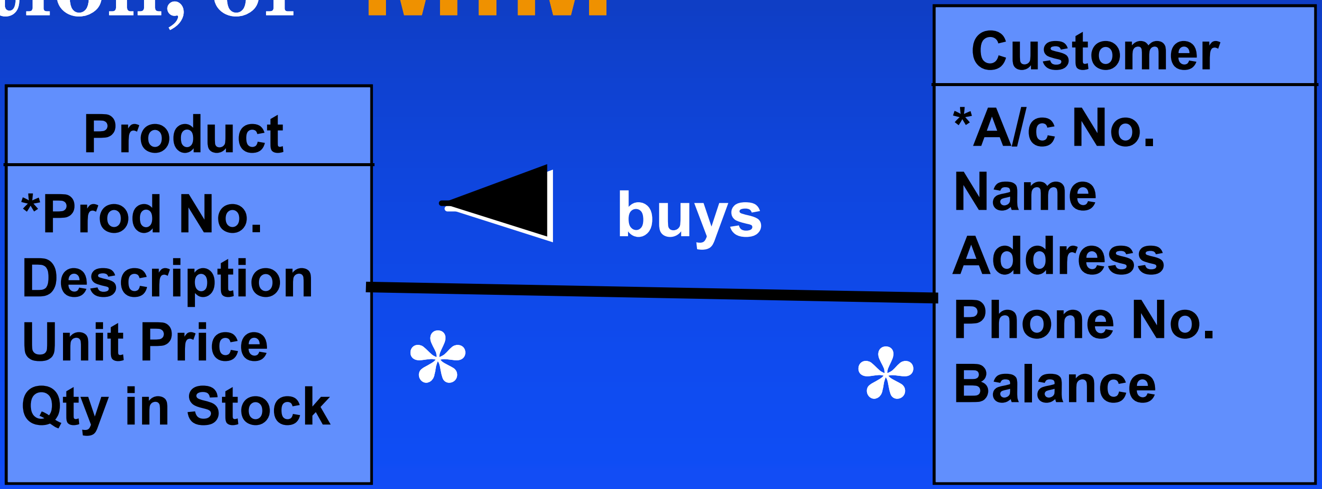
Course Offering

*Course No.
*Date
Room No.
Max Enrol

❻ The Entity-Relationship **Notation.**

Multiplicity

- And, since we wish **many** Customers to buy **many** Products,
- This one is a **Many-to-Many** association, or **M:M**



❻ The Entity-Relationship Notation. Multiplicity

- There are numerous other symbols also used for the “many” end of a relationship,
- including single and double arrows, etc.
- The star (asterisk) is the UML standard.
- Other ERD notations have their own ways to show “many” and what direction to read the sentence.

⑥ The Entity-Relationship Notation. Summary

- There are many ERD notations. We are using UML since it has become the official world standard object notation.
- An entity is a square box, with a singular name above a horizontal line.
- The primary key attribute is below the line, with an asterisk (*)
- Other attributes are listed below or on another page.

ctd. . .

❻ The Entity-Relationship Notation. Summary

- An **association** is a line joining two entities.
- Write the **verb** above the line, with arrowhead.
- Reading **in the direction of the arrow**,

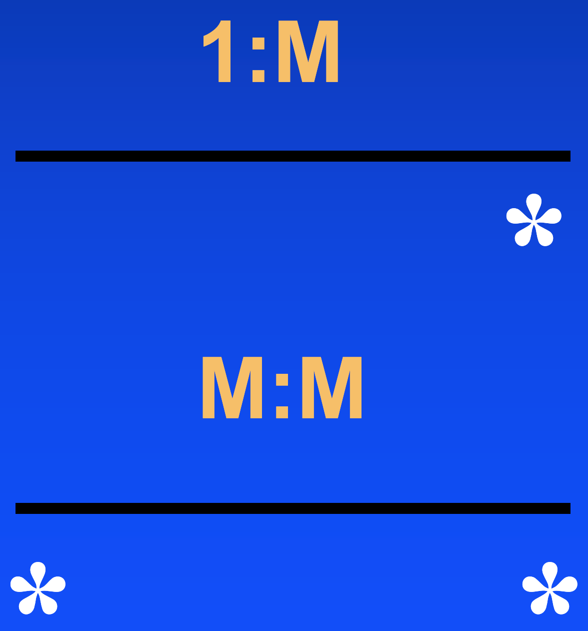
entity-verb-entity

should make a simple sentence.

ctd. . .

⑥ The Entity-Relationship Notation. Summary

- **Multiplicity** is *one* (straight line) or *many* (asterisk) below each end of the line.



4.3. Object-Oriented Models

- Object-Oriented **Programming (OOP)** is now the state of the art.
  - Pool of C programmers → C++
  - Everybody now learns Java
- Object-Oriented **Analysis and Design (OOA&D)** are still very much the leading edge.
- **Object-Oriented Databases (OODBMS)** are powerful and rapidly catching up.
- RDBMSs are going O-O.

4.3. Object-Oriented Models

- Object-Oriented **Development Environments** so far are mostly ***Object-Oriented Front Ends*** that link to a relational database.
- They are truly O-O only in the GUI.
- They are evolving toward true O-O,
- And so are the relational databases!
  - ORACLE 9i permits O-O.

4.4. An Introduction to Objects

- Like an entity, an object is described by a noun,
- And it is "Some**thing** in the user's world that has a separate and distinct existence, and is of interest in that they need to keep data **about** it."

4.4. An Introduction to Objects

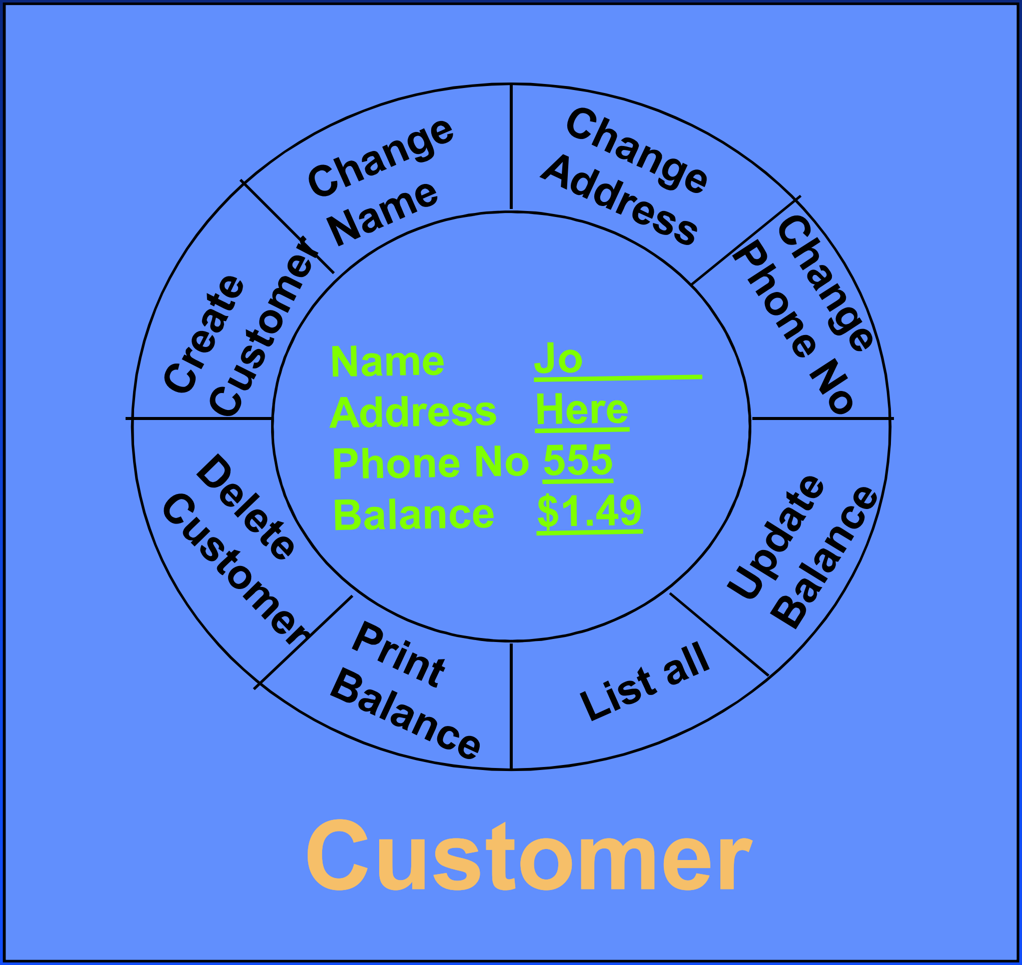
- **But an object is more than an entity.**
- **In addition to the data, the object carries program code,**
- **That *uses or changes that data.***

4.4. An Introduction to Objects

The ***only*** way that a program
can **read or change**
the data carried by an object,
or **access the data** in any way at all
is by invoking one of the defined
pieces of program code
that the object carries within itself.

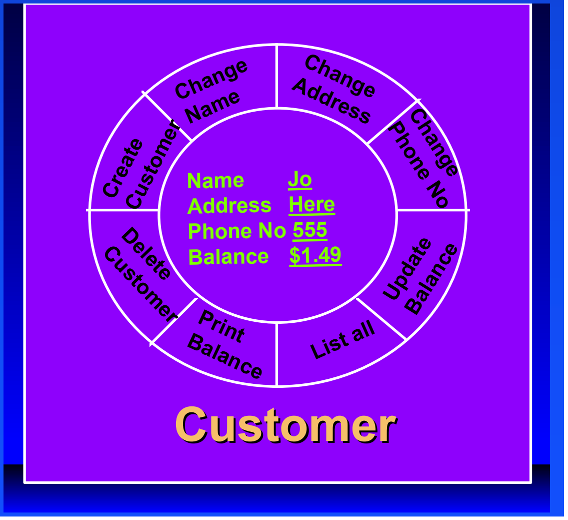
4.4. An Introduction to Objects

This can be illustrated with a Taylor Donut Diagram:



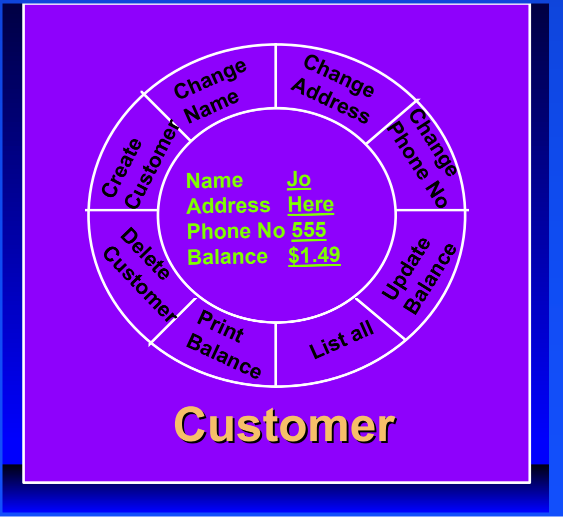
4.4. An Introduction to Objects

- The “behaviors” around the outside are **subroutines or functions.**
- They physically are pieces of **compiled program code.**
- In the real world they correspond to things this object can **do** or **have done to it.**



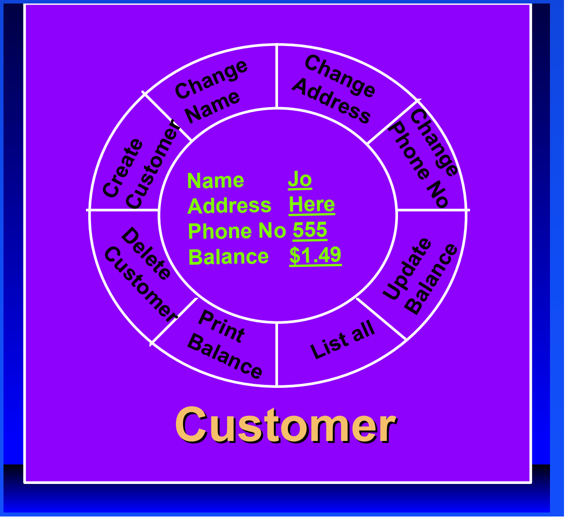
4.4. An Introduction to Objects

- These are called variously:
  - Functions
  - Behaviors
  - Operations
  - Services
  - Responsibilities
  - Methods (esp. at the OOP level)
- The data in the center is surrounded and protected by them.



4.4. An Introduction to Objects

- Thus, as Coad and Yourdon put it,
  - An object encapsulates both "the data, and the exclusive functions on that data."
  - By "exclusive" we mean that *no other code* can access or manipulate this data.



4.4. An Introduction to Objects

- So our task as Analysts is to investigate the users' business to find:
  - The objects and their classes
  - Their attributes and associations
  - performed on, to, with or by

each operation will use

ributes.

we produce

ch operation.

ese specifications to

ds in whatever OOPL

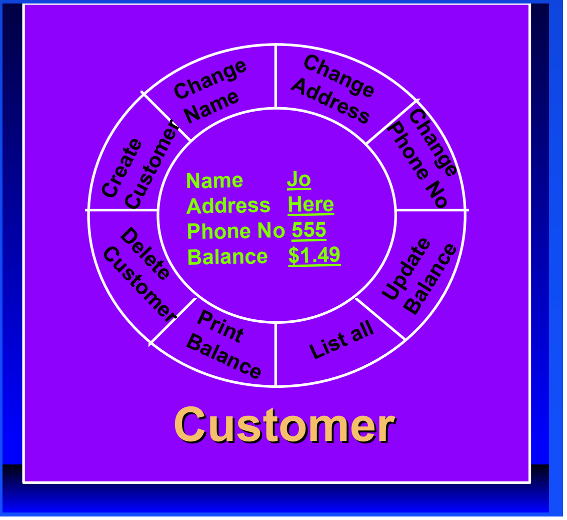
TUGAS

- Analisa Objek dan asosiasinya dari unit bisnis Yang anda interview
- Analisa menggunakan UML class diagram
- kirim ke ocal_sophan@yahoo.com paling lambat 5 hari

4.4. An Introduction to Objects

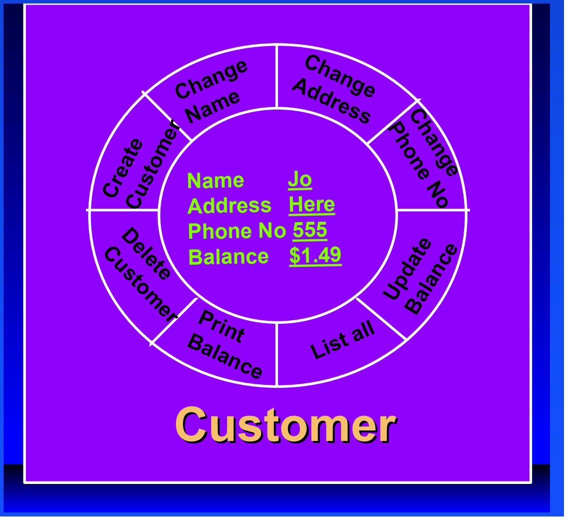
- OOPLs and OODBMSs actually store the program code for the methods in the database along with the data.
- We write small pieces of code, each to do a single operation.
- A Customer object will need:
    - An operation called UpdateAddress
    - A piece of code to carry it out

This code is then attached to the object in the database.



4.4. An Introduction to Objects

- When an object is first created, it appears as a collection of fields (attributes) *in RAM.*
- Some are **deleted** after a short lifetime.
- Some **disappear** when the machine is turned off.
- These are all **Transient Objects,** which ***DO NOT*** survive beyond the current session.

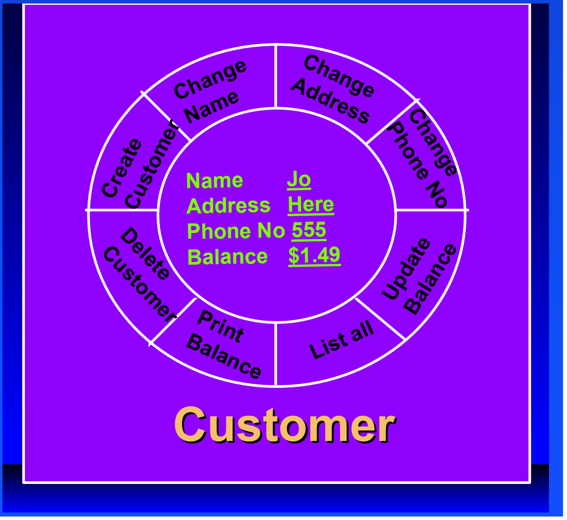


4.4. An Introduction to Objects

- Some objects we need to keep around longer
  - Customers
  - Employees
  - Products, etc.
- i.e., all the things we normally would expect to put in a *database*.
- These we create as

Persistent Objects,

that ***DO*** survive beyond the current session.



4.4. An Introduction to Objects
Summary

- Objects are entities (*things* that carry data) with behavior (operations) added.
- These operations exclusively access the data it carries - no other code can touch it.
- Operations are coded in an OOPL or OODBMS as *methods* (functions, subroutines).
- Code for the methods is stored in the database along with the data for the object.
- Any program that wants data from this object can get it only by calling one of the methods defined on this object.

4.4. An Introduction to Objects

Summary

- **Transient Objects** are created by OOPLs in the RAM of the computer and

  DO NOT Survive

  the current session.

- **Persistent Objects** are stored by a database (OODBMS) and

  DO Survive

  beyond the current session.

4.5. Object-Oriented vs Object-Based

- Some programming languages have objects but do not qualify as Object-Oriented.
  - e.g. ADA 85, Clipper
- To be truly O-O, they must have two important features:
  - **Inheritance,** and
  - **Polymorphism**

(These are defined and fully explained in Chapter 8)

- These two features account for a great deal of the power and benefits of O-O methods.

4.6. Conceptual, Logical & Physical Models

- There are a variety of modeling paradigms available: DFDs, ERDs, Objects, etc.
- Each can be done at three levels.
- These are three different levels of abstraction:

Conceptual

Logical

Physical

4.6. Conceptual, Logical & Physical Models

A *physical model* is the final design document showing how things will be written, done or built, and depicting all hardware and software platform details, including data storage and data transmission media.

4.6. Conceptual, Logical & Physical Models

At the **logical** level, we build a model that shows everything that must be included, and everything that the system must do, ***without*** specifying **how.**

It makes no reference to choices of hardware, software or media.

A ***logical model*** shows what a system must do or have, without regard for *how* it is to be done, built or represented.

4.6. Conceptual, Logical & Physical Models

A ***Conceptual Model*** is a representation of the users’ business in terms of **their conception** of how it operates.

For ERDs and Object models, this means it includes M:M associations.

4.6. Conceptual, Logical & Physical Models

- We begin by working with the users to produce a **conceptual model.**
- Then we expand that into a **logical model** by adding all the features that were not apparent to the users.
- Finally we make all our design decisions and document them on the **physical model.**

4.7. Models as Communication Tools

- Users always accuse us “technoids” of speaking in a foreign tongue.
- They complain that:
  - “Those computer people never listen to us,”
  - or “They never do what we were asking for”
  - or “They gave us way more than we needed”
  - or “They can never explain in English how to make it work.”

4.7. Models as Communication Tools

- To build an accurate model, it is essential that you are **user-driven** in your modeling.
- The difference between a *good* systems analyst and a *great* one is in **people skills,** especially **listening skills.**
- **"God gave us two ears and one mouth!"**
- Remember, we are using our object-oriented techniques to understand **their** business.

4.7. Models as Communication Tools

- Once the users learn the notation, they quickly take to using the model to discuss and solve problems with the analysts.

- All players on the team and sometimes outside it can make use of these models.

End of Chapter 4