OO Fundamentals

Mindtree

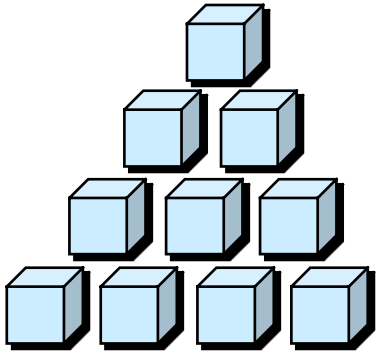
PRATIAN TECHNOLOGIES

Topics

- What is OOP?
- OOP v/s Structured Programming
- UML
  - Introduction
  - Relationships
- Classes & Objects

The procedural approach

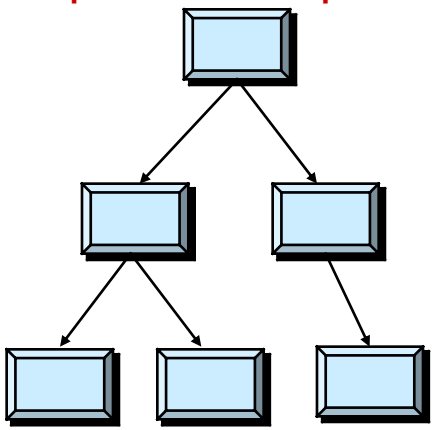
- The procedural approach
  - Deals with functions as the **building blocks**
  - Easy to start with
  - Higher comfort level for a new programmer



Functions are the building blocks

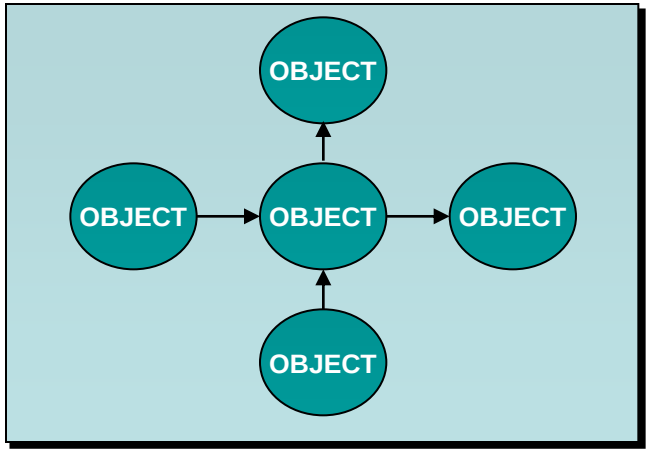
- Simple decomposition technique for
  - Modularity
  - Reusability
  - Complexity

Top – down decomposition

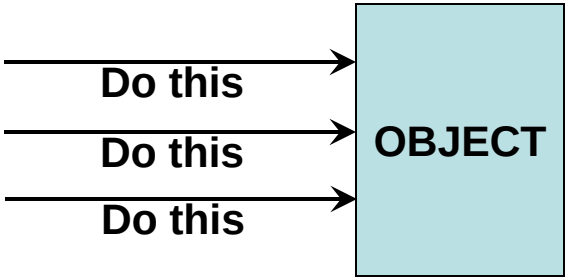


OO Approach

- An Object oriented approach views systems and programs as a collection of interacting objects.



- An object is a thing in a computer system that is capable of responding to messages



What are Objects?

- We interact with *objects* everyday
  - A customer
  - An order
  - Your car
  - The telephone

- An object represents an entity – physical, conceptual or software
  - **Physical entity**
    - Employee, Customer, Supplier
  - **Conceptual entity**
    - Sales, Policy, TaxCalculator
  - **Software entity**
    - Linked List, Connection, etc.
- ***A programmer should make a good effort to capture the conceptual entities in addition to physical entities which are relatively straight forward to identify***

Why choose the OO approach?

CustomerDet.c - Microsoft Visual Studio

File Edit View Debug Tools Window Community Help

CustomerDet.c | Start Page

```
#include "stdio.h"
int cust_id;
int product_id;
/* scores of other variables here*/
char* prod_name;
char* cust_name;
/* scores of functions here*/
void makePurchase(){.............}
/* scores of functions here*/
void changePrice(){...........}
/* scores of functions here*/
```

Can you identify where is Customer ?

Can you identify where is Product ?

Solution Explorer | Class View

Ready | Ln 1 | Col 11 | Ch 11 | INS

Why choose the OO approach ?

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace OrderProcessing
{
    public class Customer
    {
        public int Id;
        public string Name;
    }
}
```

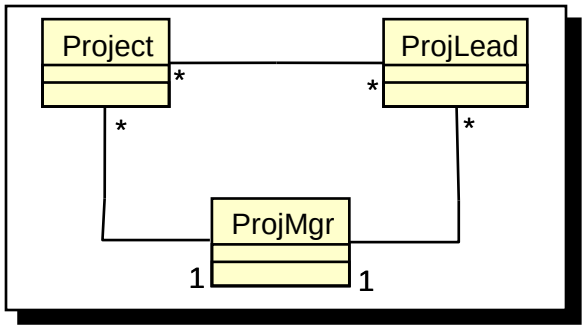
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace OrderProcessing
{
    public class Product
    {
        public int Id;
        public string Name;
    }
}
```

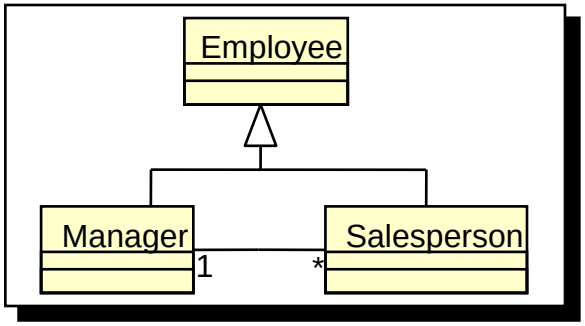
Easier to comprehend

Why choose the OO approach?

- The OO approach
  - Deals with classes as the building blocks
  - Allows Real World Modeling
  - The idea of OOP is to try to approach programming in a more natural way by grouping all the code that belongs to a particular object— such as an account or a customer — together



- Raise the level of abstraction
  - Applications can be implemented in the same terms in which they are described by users
- Easier to find nouns and construct a system centered around the nouns than actions in isolation



- Easier to visualize an encapsulated representation of data and responsibilities of entities present in the domain
- The modern methodologies recommend the object-oriented approach even for applications developed in C or Cobol

Object-Oriented Programming

"Object-oriented programming is a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class..."

Grady Booch

Procedural vs. Object-Oriented Programming

- The unit in procedural programming is *function*, and unit in object-oriented programming is *class*
- Procedural programming concentrates on creating functions, while object-oriented programming starts from isolating the classes, and then look for the methods inside them.
- Procedural programming separates the data of the program from the operations that manipulate the data, while object-oriented programming focus on both of them

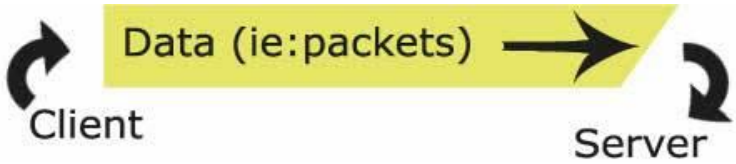


figure1: procedural

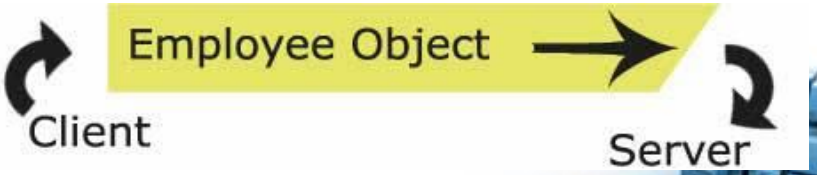
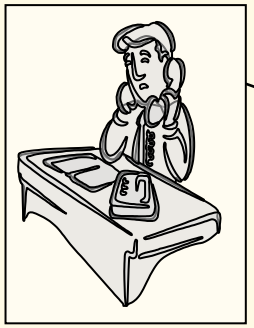


figure2: object-oriented

What is a Class?

A **class** gives a generic notion of what objects look like



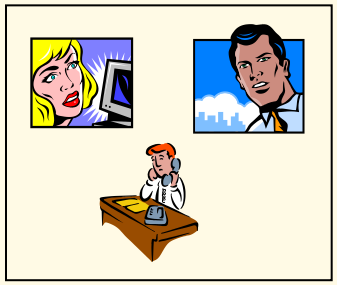
Employee
Class

Instantiate (create)

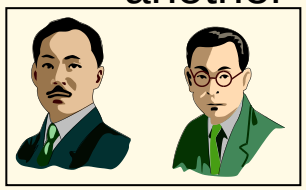
An employee
Joe

An **object** is an instance of a class

Classes are used to distinguish one type of object from another



Employee



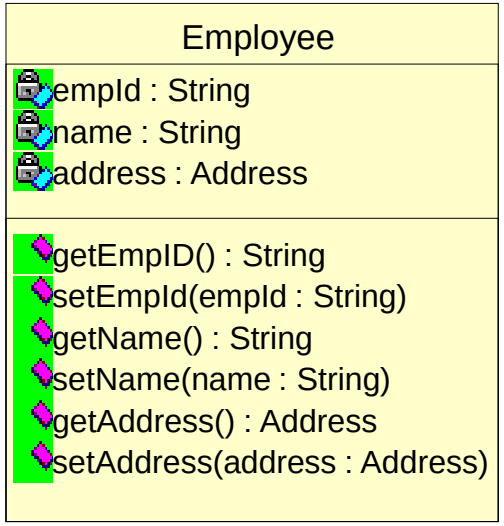
Reg. Customer

Customer

Types

Class

- User defined type
  - Encapsulates all the data and operations pertaining to an entity
  - Provides a Single representation to all the attributes defining the entity
  - Passing single representations is easier



- Data types as collections
  - A struct in C encapsulates only data. Used as a data structure to store different types of data
  - An array is used to store different elements of the same type

Relationship between Classes

- Classification

<<Is-a>>	<<Has-a>>	<<Uses>>
Generalization Realization	Association Aggregation Composition	Dependency

- For all practical purposes we will represent
  - Is-a relationship as

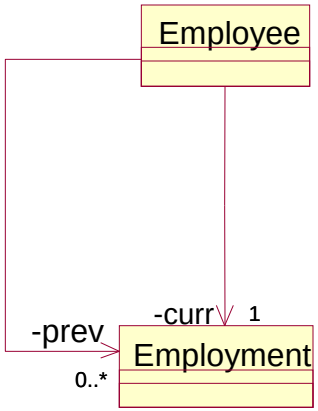
    ——————▶
  - Has-a relationship as

    ————————
  - Uses relationship as

    ---------->

Has-a Relationship

- The 'Has-a' relationships are qualified by
  - Multiplicity
    - The number of instances with which a class is associated
    - Can be 1, 0..1, *, 1..*, 0..*, 2..*, 5..10, etc.
    - Multiplicity is by default 1
  - Navigability
    - Can be unidirectional or bidirectional
    - Navigability is by default bi-directional
  - Role name
    - The name of the instance in the relationship
    - Multiple 'has-a' based on different roles are possible



Identifying Classes

A trainer trains many trainees on a given technology in this course, which contains many modules – each module is comprised of different units and each unit has many topics.

- Identify the different classes from the above problem statement

Procedural approach

- Focus is on identifying **VERBS**
- Connections between functions established through Function Calls

OO approach

- Focus is on identifying **NOUNS**
- Connections between classes established through Relationships ('Is-a' and 'Has-a')

Identifying Classes

- Trainer
- Trainee
- Course
- Technology
- Module
- Unit
- Topic

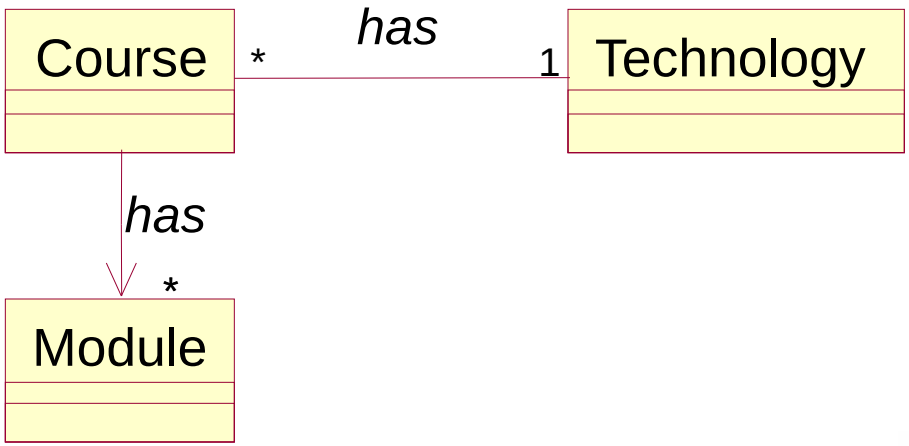
- Identify the different connections (relationships) between the above classes

Identifying Relationships

- Trainer - Trainee
  - Trainer ‘HAS’ many Trainees
  - Every Trainee ‘HAS’ a Trainer

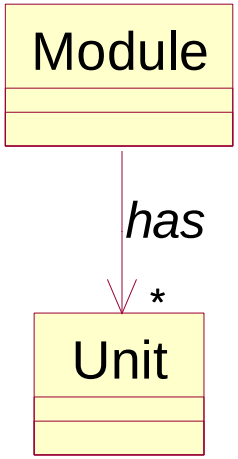
Identifying Relationships

- Course – Technology
- Course - Module
  - Course ‘HAS’ an associated Technology
  - A Technology has many courses
  - Course ‘HAS’ many Modules

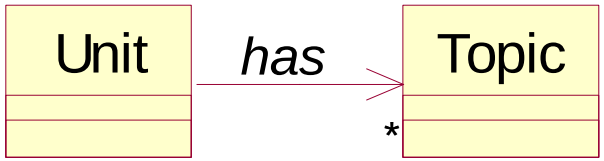


Identifying Relationships

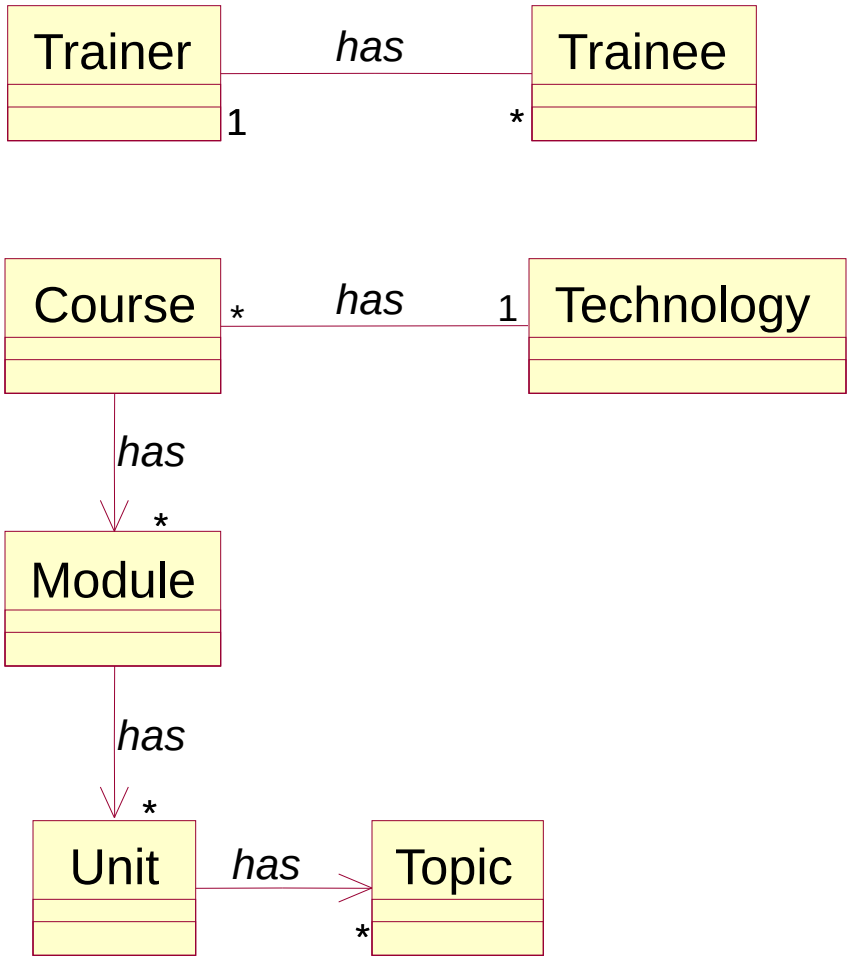
- Module – Unit
  - Module ‘HAS’ many Units



- Unit – Topic
  - Unit ‘HAS’ many Topics



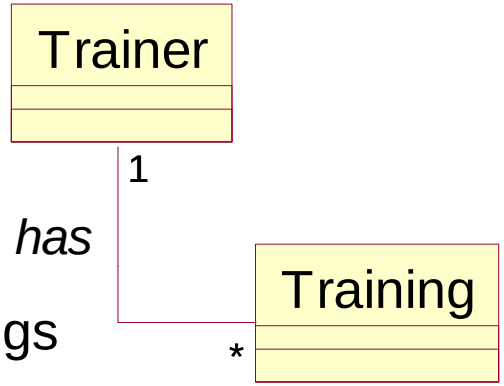
The OO Model



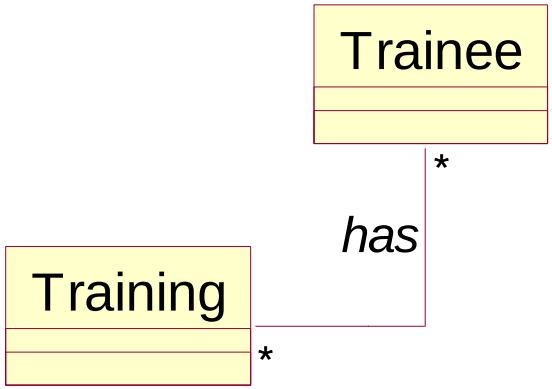
- How do you relate the Trainer & Trainee to the Course?

Conceptual Entity

- Trainer – Training
  - A Trainer (HAS) conducts many Trainings
  - A Training HAS a Trainer

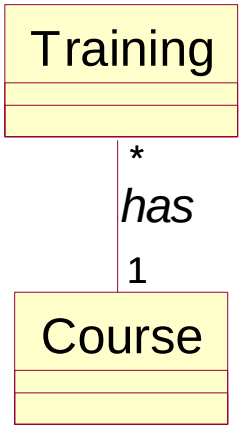


- Trainee – Training
  - A Trainee (HAS) attends many Trainings
  - A Training HAS a many Trainees

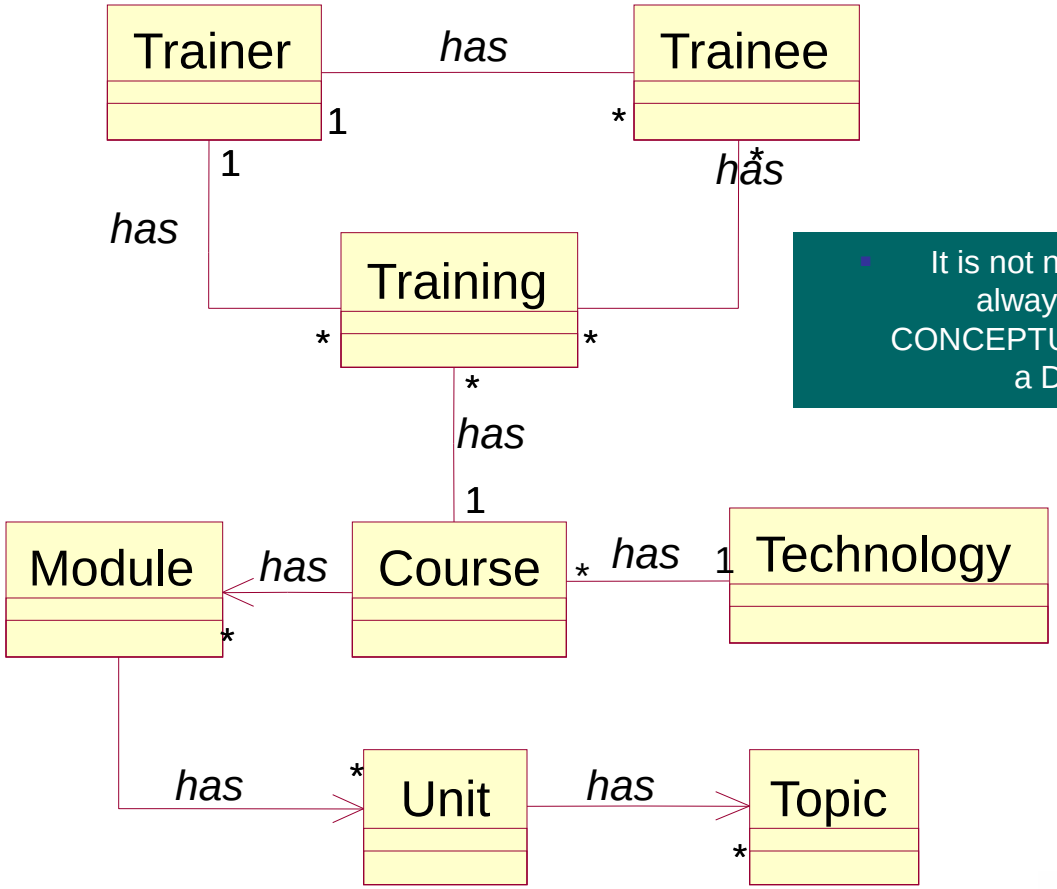


Conceptual Entity

- Training - Course
  - The Training (HAS) an association with a Course (conducted for a Course)
  - A Course HAS many Trainings



Solution



- It is not necessary to always have a CONCEPTUAL ENTITY in a Design

- Easier to model real-world problems through the OO approach than through the procedural approach

Exercise

A company sells different items to customers who have placed orders. An order can be placed for several items. However, a company gives special discounts to its registered customers.

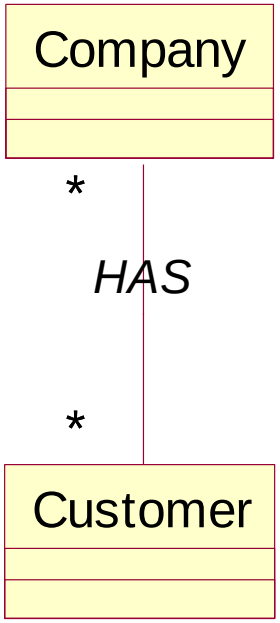
- Identify the different classes from the above problem statement
- Identify the different connections (relationships) between the above classes

Identifying Classes

- Company
- Item
- Order
- Customer
- RegCustomer

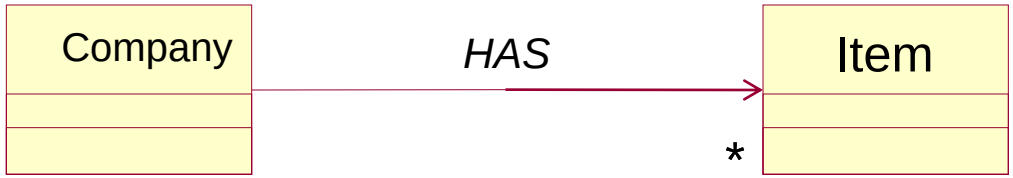
Identifying Relationships

- Company - Customer
  - Company ‘HAS’ many Customers
  - Customer ‘HAS’ many Companies



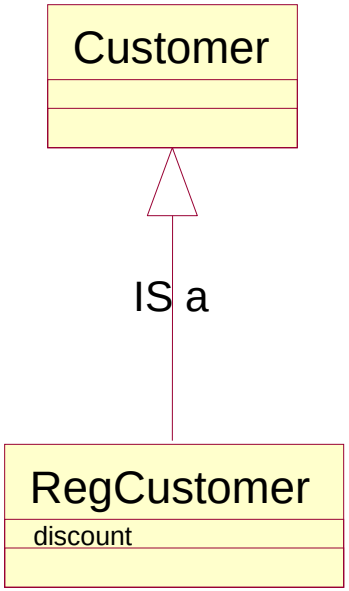
Identifying Relationships

- Company - Item
  - Company HAS many Items



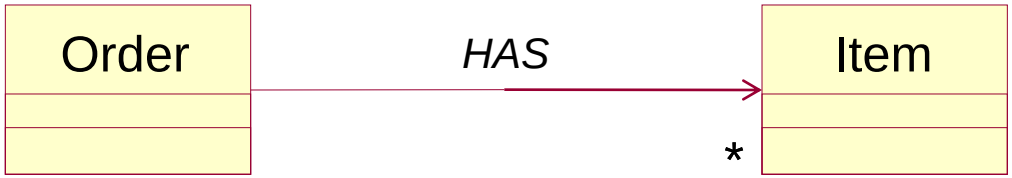
Identifying Relationships

- Customer - RegCustomer
  - RegCustomer ‘IS’ a Customer



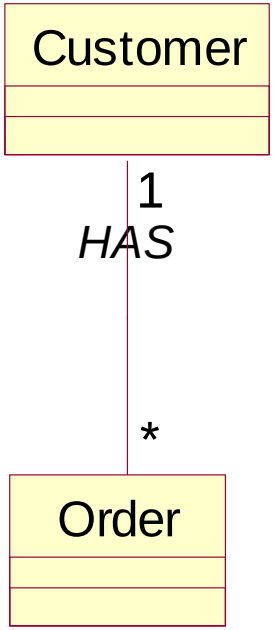
Identifying Relationships

- Order - Item
  - Order HAS many Items

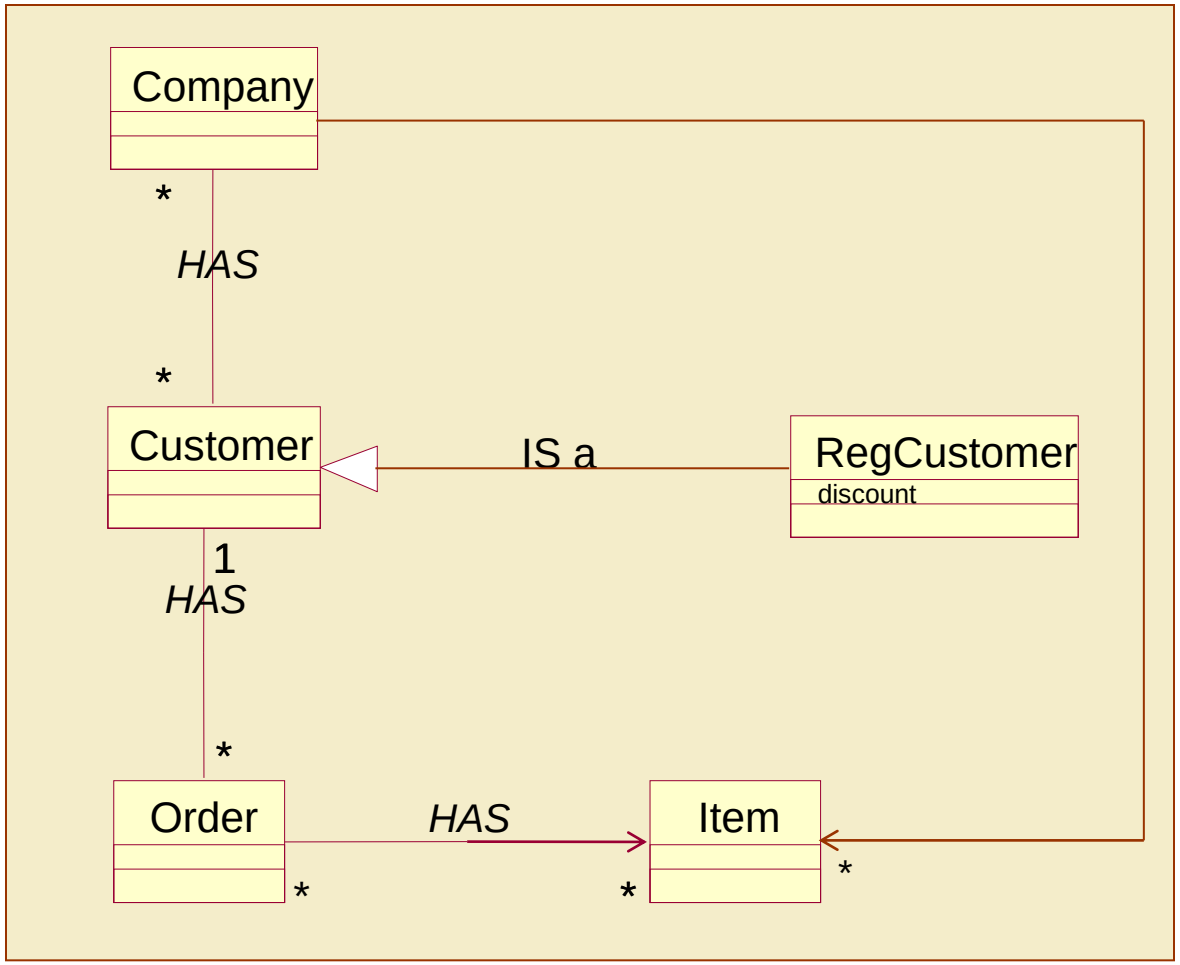


Identifying Relationships

- Customer - Order
  - Customer HAS many Orders
  - Order HAS one Customer



The OO Model



A Customer can place many orders implies that RegCustomer can also place many Orders.

A Company has many Customers implies that a Company also has many RegCustomers

Exercise

In the SkillAssure Assessment Framework ,

Every course can have assessments

An Iteration has many courses and can also have additional assessments

The training model has 4 Iterations

An assessment can be of multiple-choice type,

hands-on exercise or project

- Identify the different classes from the above problem statement
- Identify the different connections (relationships) between the above classes

Exercise

There are many programming languages. Java and C# are object-oriented programming languages. C is a procedural programming language.

- Identify the different classes from the above problem statement
- Identify the different connections (relationships) between the above classes

Class in C#

- A *class* is a software construct that defines the instance variables and methods of an object.
- A *class* is a template that defines how an object will look and behave when the object is created or instantiated from the specification declared by the class.
- A *class* can be viewed as a user defined data type.

```
class Point
{
        double x;
        double y;

        public double GetValue()
        {
                return x * y;
        }
}
```

Structure of a class

```
public class Employee
{
        String employeeId;
        String employeeName;

        Employee()
        {
                Console.WriteLine("Constructor called");
        }

        public String EmployeeId {get; set;}
        public String EmployeeName {get; set;}


        public void DisplayEmployee()
        {
                Console.WriteLine("Emp Id : " + EmployeeId);
                Console.WriteLine("Emp Id : " + EmployeeName);
         }
```

Instance Variables

Constructor

Properties

Methods

What is an Object?

- An object is an entity with a well-defined *State* and *Behavior*
- An *object* is created from the class definition using the *new* operator.
- The state of an object is referred to as the values held inside the instance *variables* of the class.
- The behavior of the class is referred to as the *methods* of the class.
- To create an object of the class Point, say,
  Point p = new Point();

```
class Point
{
        double x;
        double y;

        double GetX()
        {
                return x;
        }
}
```

- When an object of the class is created, memory is allocated for all the instance variables, here p is not an object but a *reference* or *handle* to an object being created.

Instantiating Classes

```
public class Shop
    {
        Product p1=new Product();
        p1.id=1;
        p1.name="Steam Iron";

        Product p2=new Product();
        p2.id=2;
        p2.name="Microwave"

        p1.makePurchase();
        ................
    }
}
```

P1 is a reference

The RHS creates an instance

Modeling the 'has-a' relationship


```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace OrderProcessing
{
    public class Trainee
    {
        public int Id;
        public string Name;

        public Trainer _Trainer;
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace OrderProcessing
{
    public class Trainer
    {
        public int Id;
        public string Name;
    }
}
```

Modeling 'has-a' with multiplicity 'n'


```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace OrderProcessing
{
    public class Trainer
    {
        public int Id;
        public string Name;

        private Trainer[] Trainees = new Trainer[20];
    }
}
```

Comparing Objects

- The '==' operator when used with objects, does not compare the states of the objects.
- Instead it compares whether the two references point to same object in memory or not.
- It is simply because the compiler does not know how to compare user defined types, Eg., how can the compiler know how to compare 2 customers (i.e., objects of Customer class)
- To do more meaningful comparison, the equals method is used.
  - The programmer is responsible for providing this method for his classes

Question time

Please try to limit the questions to the topics discussed during the session. Thank you.