Programming I

Course 12

Introduction to programming

What we talked about?

- Relation between classes
    - Has a
    - A kinf of
    - Is a

- Inheritance

What we talk today?

- Object Oriented Analyze/Design/Programming
- How to represent classes?
  - Code
  - Graphical notation
  - Informal notation

What to do when a problem is enounced?

- Identify problem
  - Input
  - Output

- How to identify?
  - Modeling the problem

- What not to do?
  - Rush to the code

Way not to rush to the code?

- Cannot design a solution if the requirements are not understood

- One cannot implement the design if the design is faulty.

- Analyze different alternatives to resolve the problem

- Critical ability to develop in OO is to think in terms of objects and to artfully assign responsibilities to software objects.

What to do?

- Analysis
  - Investigate the problem and the requirements.
  - What is needed? Required functions? Investigate domain objects.
  - The What's of a system.

- Design
  - Conceptual solution that meets requirements.
  - Not an implementation
  - Avoid commonly understood functionality (constructors, set/get methods, …).
  - The How's of the system

Formalize the previous discussions

- Object Oriented Analyze – OOA
  - find and describe objects or concepts in the problem domain
- Object Oriented Design – OOD
  - define how these software objects collaborate to meet the requirements.
    - Attributes and methods.
- Object Oriented Programming – OOP
  - Implementation
    - Different OO languages

Object Oriented Analyze

- Goal
  - To model the problem domain by developing an object oriented system
- Input
  - Problem requirements
  - Specifications (can include use case diagrams or other types of diagrams)
- Output
  - Conceptual model
  - Uses case
  - Any other documentation

Object Oriented Analyze

- Does not take into account implementations details (database structure, persistence model) this are described by OOD

- Graphical notations
  - Coad, Yourdon, Rumbaugh, Booch, Firesmith, Embley, Kurtz, etc
  - Unified Modeling Language (www.uml.org) (UML) – standard for OOA

- Tasks of OOA
  - Identifying the objects
  - Identifying relations between objects
  - Define use cases
  - Define user interface (UI)

Object Oriented Design

- Goal
  - To define(refine) the objects, the object interaction and the documents identified at object oriented analyze step
- Makes the transition from software architecture to software development
- Input
  - OOA output (conceptual model, use case diagram, UI documentation, others documents)
- Output
  - Class diagrams
    - Describe classes (attributes & methods) and interaction between them (inheritance, dependence, association, composition)
  - Sequence diagrams
    - Message flow (communication) between objects

Object Oriented Design

- Steps
  1. Object definition: attributes, behavior, exposed services
  2. Developing diagrams from conceptual model
  3. Identify application framework
     - Identify a set of library or classes in order to structure the application
     - Reduce the developing time by reusage of implemented functionalities
  4. Identify persistent objects/data (data that is stored)
  5. Identification & definition of remote objects
  6. Evaluation of OO languages and choosing the appropriate one
  7. Evaluate OO design
  8. Define testing strategies
     - Unit testing, integrations test, regression testing, etc

How to do this?

- Through **experience** and common sense
- Using OOD **principles**, design patterns

Objects Attributes

- Finding attributes
  - Use first person
  - Problem analyze, address questions to client

- Identify attribute definition domain

- Identify the relation between attribute

- Example
  - A person has like attributes height
    - Should be positive and less than 3 meters

Structuring objects

- Generalization/specialization (identify hierarchies)
    - Use inheritance to group common attributes and behavior
    - The reunion of all specializations covers the hole generalization?
    - The specializations are exclude each other
    - Example
        - Figure, Circle, Line

- Hole-part relations (has a)
    - The hole does not inherit the behavior from the parts => the inheritance is not applicable
    - Example
        - Line, Polygonal Line

Objects services

- Member functions
    - Implicit services
        - New instances creation, set/get methods
    - Services associated with messages
        - Identify messages sent to objects
    - Services associated with objects relations
        - Example: A polygon has multiple points => add/remove points from it
    - Services associated with attributes
        - Protect some attributes, real time synchronization

What we talk today?

- Object Oriented Analyze/Design/Programming

- How to represent classes?
  - Code

  - Graphical notation

  - Informal notation

Graphical Representation of objects

- Most accepted standard
    - UML (Unified Modeling Language)
- Types of diagrams
    - Behavior – describe the behavior of the system or business process
    - Interaction – more detailed diagrams for system behavior
    - Structural diagrams – describe in detail the specifications that are transparent at design step
        - Class diagram
        - Object diagrams

Class Diagrams

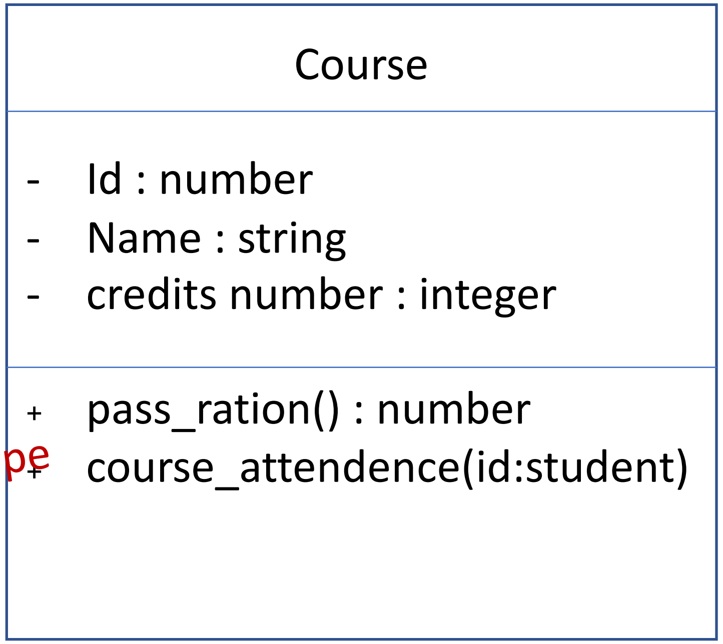
- Graphical representation of classes and class relations

- Class is represented like a rectangle that has three parts
    - Class name
    - Class attributes

      Syntax: visibility attribute_name : attribute_type
    - Class methods

      Syntax: visibility method_name (parameter:parametr_type) : return_value_type

Private (-)
Public (+)
Protected (#)
Package (~)



Course

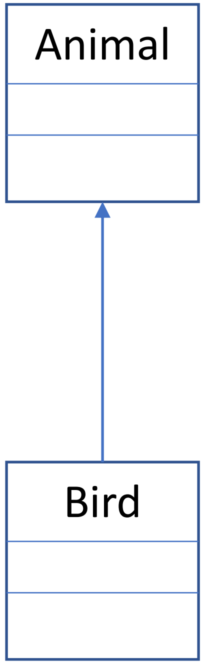
- Id : number
- Name : string
- credits number : integer

+ pass_ration() : number
+ course_attendence(id:student)

The type does not have to be linked to the exact name of a programming language data type

Class Diagrams

- Relations
  - Inheritance
    - A bird is a kind of animal
    - Graphical representation
      - A arrow that points to super class
  - Dependency
  - Association /Aggregation
  - Composition



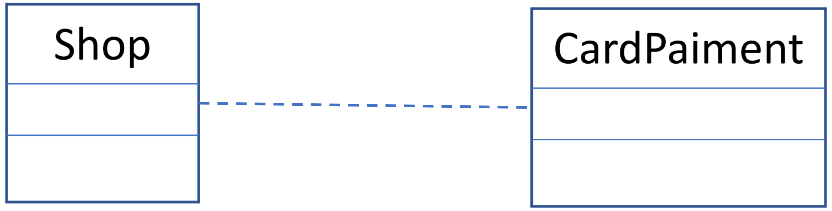
Code

```
class Animal(object):
    def __init__(self):
        print("Animal")

class Bird(Animal):
    def __init__(self):
        Animal.__init__(self)
        print("Bird")
```

Class Diagrams

- Relations
  - Inheritance
  - Dependency
    - A Shop uses Card Payment
    - Graphical representation
      - A dashed line (can have a arrow starting from the dependent class to it)
  - Association /Aggregation
  - Composition



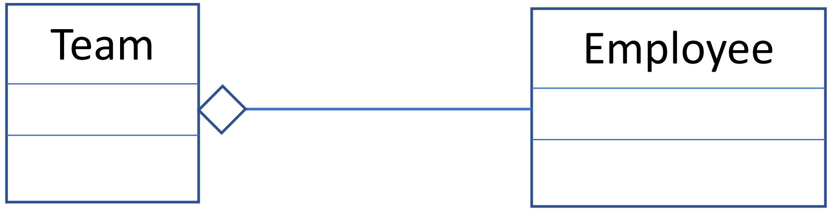
Code

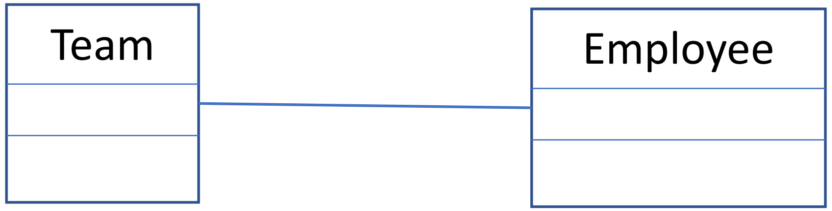
```
class Shop(object):
   def __init__(self):
      print("Shop")
   def pay_with_card(self, amount, banck,
                                  card_id):
       # …
       cp = CardPayment(bank, card_id)
       cp.pay(amount)
       # …

class CardPayment(object):
   def __init__(self):
       print("Card payment")
```

Class Diagrams

- Relations
    - Inheritance
    - Dependency
    - Association/Aggregation
        - A Team has a list of Employee
        - Graphical representation
            - Fill line
    - Composition



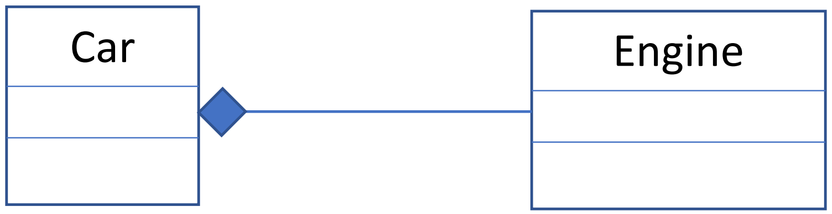


Code

```
class Employee(object):
   def __init__(self, name):
      print("Employee")

class Team(object):
   def __init__(self):
       print("Team")
       self.list_employees = []
   def addEmployee(self, emp):
       self.list_employees.append(emp)
```

Class Diagrams



- Relations
  - Inheritance
  - Dependency
  - Association/Aggregation
  - Composition
    - An Engine is a part of a Car
    - Graphical representation
      - Fill line

Code

```
class Engine(object):
    def __init__(self, power, type):
        print("Engine")
        self.power = power
        self.type = type

class Car(object):
    def __init__(self, engine_power):
        print("car")
        self.egnine = Engine(engine_power,
                                         "Electric")
```