UML and Classes, Objects and Relationships [1]

Defining Domain Models Using Class Diagrams

Agenda

- Recap:
  - Phases of software design and Unified Process
  - Object Oriented Design and Techniques
- UML Notations for Modeling Classes
- Class Relationships and UML Notations
  - Association
  - Generalization
  - Realization
  - Dependency

Software Development Process and Unified Modeling Language (UML)

- A software development process is a set of phases that are followed to bring a product or a system from conception to delivery.
- In the Unified Process, there are four of these phases:
  - Inception (*Analysis phase*)
    - identify the system we are going to develop, including what it contains and its business case.
    - UML: use-case diagrams
  - Elaboration (*Design phase*):
    - perform detailed design and identify the foundation of system from "use case diagram", which eventually lead to classes.
    - UML: classes, objects, class diagrams, sequence diagram, collaboration diagrams etc.
  - Construction (*Implementation phase*): write software using Java/C++
    - the actual building of the product from the design of the system.
  - Transition (*Rolling out phase*): Deliver the system/product to the users. Includes maintenance, upgrades, and so on until phasing out.

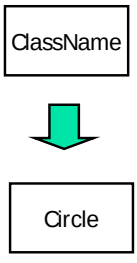
Object Oriented Design (OOD)

- OOD is the technique used to architect software with groups of classes that interact with one another to solve a problem.
- To qualify as an OOD, a few requirements need to be met.
- The three fundamental principles are essential for an OOD to exist:
  - Classes (abstraction and encapsulation)
  - Inheritance
  - Polymorphism
- + OO notions: packages, exceptions, streams, threads, components and events (asynchronous notifications), and communicators (sockets).

Modeling a class in UML

- When modeling a class in UML, we have a lot of flexibility.
- The same class can be modeled in four different ways:
  - With no attributes or operations shown
  - With only the attributes shown
  - With only the operations shown
  - With both the attributes and operations shown
- The name of the class is always the first component of the class box; in fact, it is the only required component, as we have seen in our earlier discussion (OOP in Java).

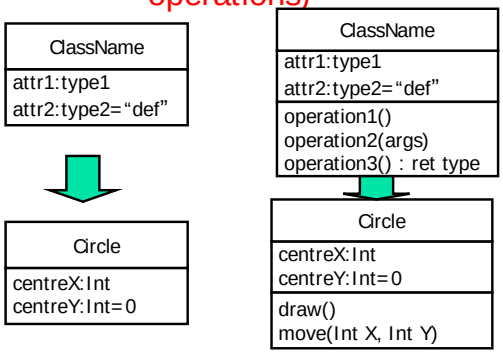
Classes (with no members)



Notes

- **Class** - The rectangle is the icon for the class. The name of the class is, by convention, a word with an initial uppercase letter. It appears near the top of the rectangle. If your class name has more than one word name, then join the words together and capitalize the first letter of the every word.

Classes (with attributes and operations)

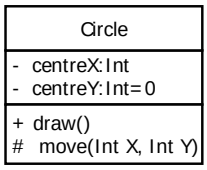


Notes

- **Attribute -** An attribute is a property of a class. It describes a range of values that the property may hold in objects of the class. A class may have zero or more attributes. A one-word attribute name is written in lowercase letter. If the name consists of more than one word, the words are joined and each word other than the first word begins with an uppercase letter. The list of attribute names begins below a line separating them from the class name.
- **Operations** : An operation is something that a class can do, or that you (or another class) can do to a class. Like an attribute name, an operation's name is written in lowercase letter. If the name consists of more than one word, the words are joined and each word except the first word begins with an uppercase letter. The list of operations begins below a line separating operations from the attributes.

Class Visibility

- public level +
- protected level #
- private level -



Notes

- **Visibility -** Visibility applies to attributes or operations, and specifies the extent to which other classes can use a given class's attributes or operations. Three levels of visibility are possible (last symbols are used in UML classes to indicate different levels of visibility):
- **public level** - usability extends to other classes +
- **protected level -** usability is open only to classes that inherit from original class #
- **private level** - only the original class can use the attribute or operation -

Class Multiplicity

- A multiplicity in a class specifies the number of instances (objects) of that class that can exist simultaneously.

- Only one Library object is allowed in the system (referred to as a *singleton* object).
- Default multiplicity is 0 or more (if not specified)

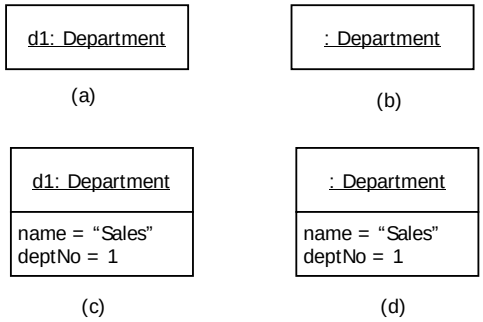
Class - Implementation

```
public class Circle {

   private int  centreX,  centreY; // centre of the circle

   //Methods to return circumference and area
   protected double move() {
     // Implementation here
   }
   public double draw() {
     // Implementation here
   }
}
```


Objects: Instances of Classes



Notes

- Figures (a)-(d) show four possible representations of objects.
  - Top –
    - objectName:ClassName – underlined or
    - :ClassName – underlined
  - Bottom –
    - Attribute names and values.

Classes, Objects, and Packages

- Packages are a way of grouping classes into common categories.
- Classes and Objects are modeled when showing the package they belong to. Classes and objects will interact with classes and objects of different packages—this is how packages are tied together to create a system.
- A package is expressed by appending the name of package and a double colon before the class name in either a class or an object.

PackageName::ClassName

ObjectName:Packagename::ClassName

Class Relationships

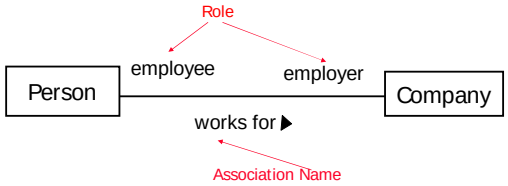
- Classes can related to each other through different relationships:
  - Association (delegation)
  - Generalization (inheritance)
  - Realization (interfaces)
  - Dependency

Association

- Association describes a link, a link being a connection among objects between classes.
- Association is shown by a solid line between classes.

Association - Example

- A Person works for a Company.



Association - Properties

- Name
  - *Name of the association*
- Role
  - *The specific role of the association*
- Multiplicity
  - *Indicates the number of objects that are connected*
- Type
  - *Plain association, aggregation, composition*
- Direction
  - *Direction can also be shown for a association*

Notes

Name : *"works for"* Is the name of the relationship.

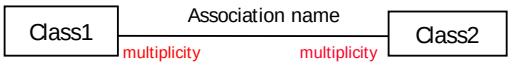
Role : *Person* plays the role *employee* and the *Company* plays the role *employer.*

Multiplicity : Many employees to one company.

Type : This shows a plain association (normally referred to as association)

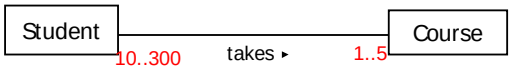
Association - Multiplicity

- Multiplicity on association specify properties of the number of links that can exist between instances (objects) of the associated classes.
  - That is, it indicates how many objects of one class relate to one object of another class. It is indicated by a single number or a range of numbers.
- We can add multiplicity on either end of class relationship by simply indicating it next to the class where the relationship enters.



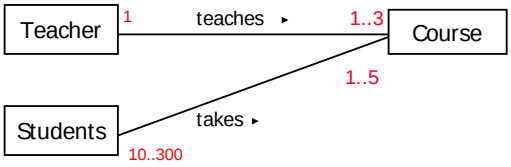
Association - Multiplicity

- A Student can take up to five Courses.
- Student has to be enrolled in at least one course.
- Up to 300 students can enroll in a course.
- A class should have at least 10 students.



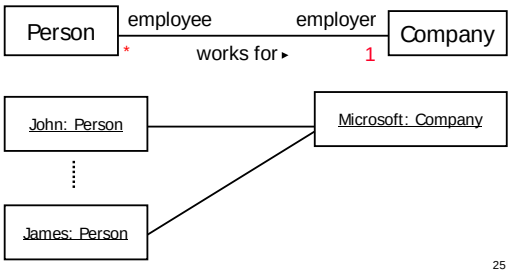
Association – Multiplicity

- A teacher teaches 1 to 3 courses (subjects)
- Each course is taught by only one teacher.
- A student can take between 1 to 5 courses.
- A course can have 10 to 300 students.



Association - Multiplicity

- Company can have many employees. An employee can only work for one company.



Association - Implementation

```
class  Company{
        Vector employee;
        public Company(){
                employee = new Vector();
        }
        public static void addEmployee(Employee emp){
                employee.addElement(emp);
        }
        public static void removeEmployee(Employee emp){
                empoyee.removeElement(emp);
        }
}
class Employee{
     ……..
}
```


Summary

- Recap:
  - Phases of software design and Unified Process
  - Object Oriented Design and Techniques
- UML Notations for Modeling Classes
- Class Relationships and UML Notations
  - Association
  - Generalization
  - Realization
  - Dependency