Object-Oriented Concept: UML Class Diagram

Shin-Jie Lee (李信杰)
Associate Professor
Computer and Network Center
Department of CSIE
National Cheng Kung University

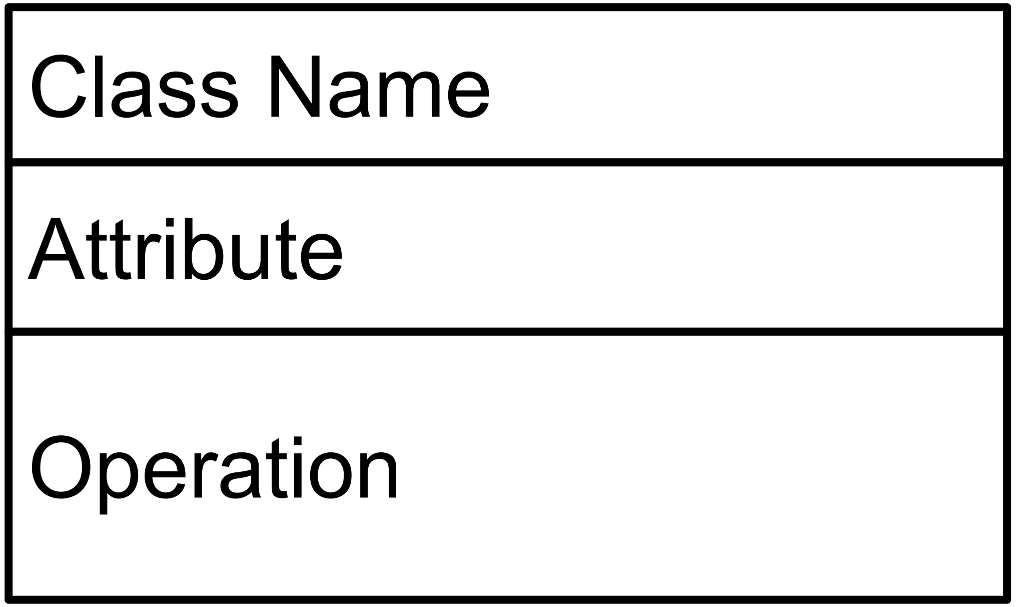
Class Diagrams

- ❑ Class is a kind of classifier.
- ❑ A Classifier represents a group of things with common properties.
- ❑ Provide a way to capture how things are put together, and make design decisions:
  - ➢ What classes hold reference to other classes.
  - ➢ What the interactions are among classes.
  - ➢ Which class owns some other class.

Class

- A class is a definition of the behavior of an object, and contains a complete description of the following:
  - The data elements (variables) the object contains
  - The operations the object can do
  - The way these variables and operations can be accessed
- *Objects are instances of classes*
- Creating instances of a class is called *instantiation.*

Class Notation



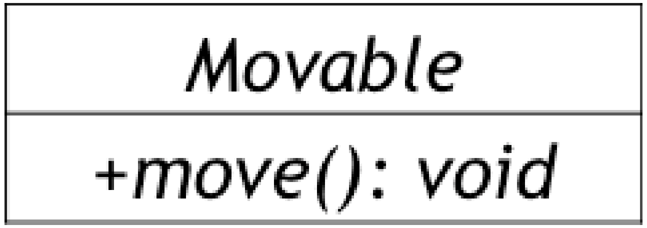
Class Name

Attribute

Operation

Abstract Class

- ❑ Abstract classes provide an operation signature but no implementation.
  - ➢ e.g.

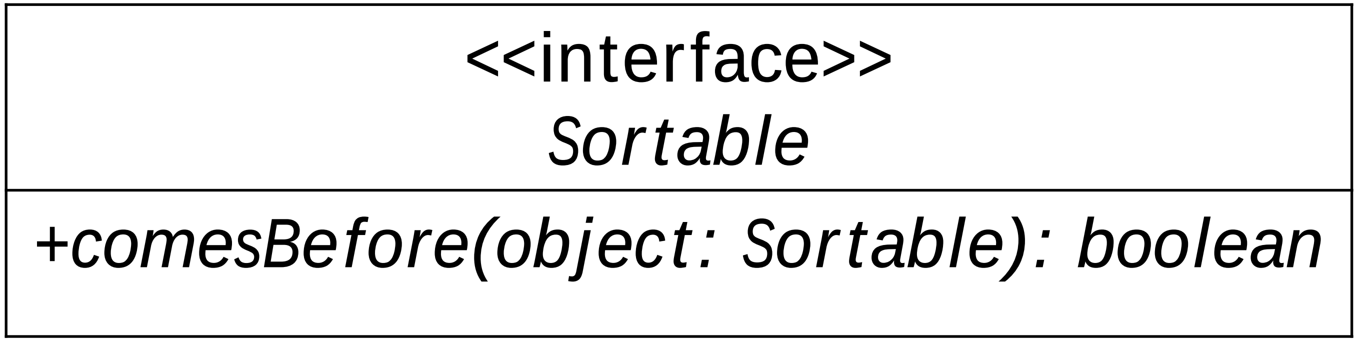


Movable

+move(): void

Interface

- ❑ An interface is a classifier that has declarations of properties and methods but no implementations.
  - ➢ e.g.



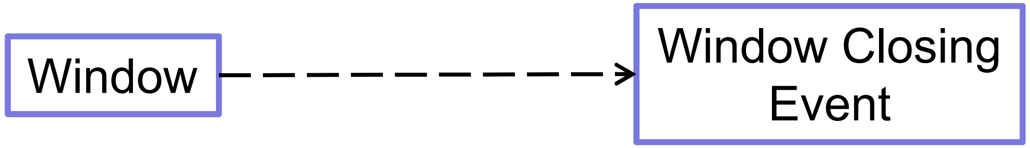
<<interface>>
Sortable

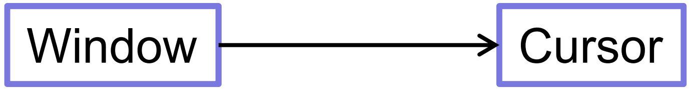
+comesBefore(object: Sortable): boolean

Inheritance

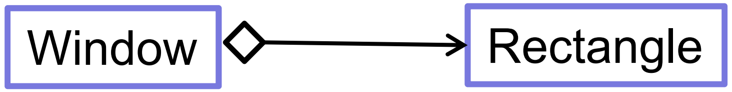
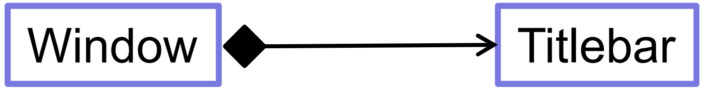
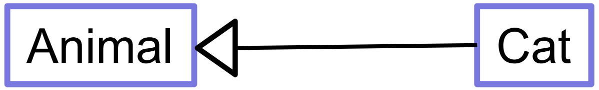
- ❑ The sharing of attributes and operations among classes based on a hierarchical relationship
- ❑ Each subclass inherits all of the properties of its superclass and adds its own unique properties (called extension)
- ❑ Facilitate reusability

Relationship1

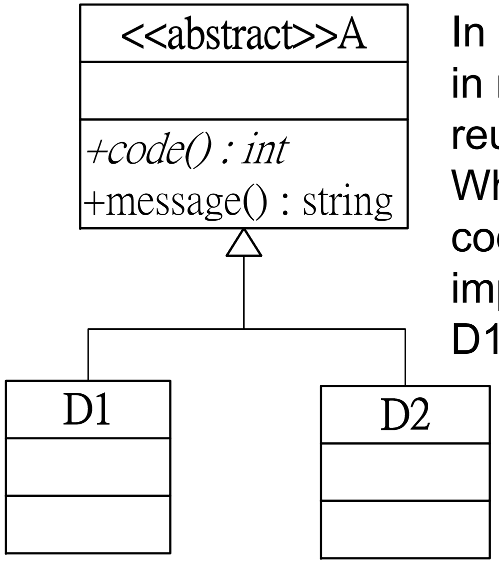
- ❑ Dependency is the weakest relationship between classes.
  - ➢ Uses-a
  - ➢ A transient relationship, that is, doesn't retain a relationship for any real length of time
  - ➢ A dependent class briefly interacts with the target class
  - ➢ 

- ❑ Association is stronger than dependency.
  - ➢ One class retains a relationship to another class over an extended period of time
  - ➢ Has-a
  - ➢ 

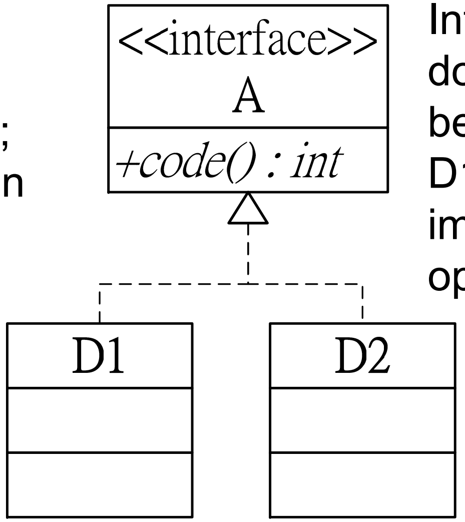
Relationship$_2$

- ❑ Aggregation is a stronger version of association.
  - ➢ **Implies** ownership
  - ➢ Owns-a
  - ➢ 
    Window
    Rectangle
- ❑ Composition represents a very strong relationship between classes to the point of containment.
  - ➢ A whole-part relationship
  - ➢ Is-part-of
  - ➢ 
    Window
    Titlebar
- ❑ Generalization
  - ➢ Is-a
  - ➢ 
    Animal
    Cat

Inheritance Example



In abstract class, code in message() can be reused in class D1 & D; While abstract operation code() needs to be implemented in D1 & D2.



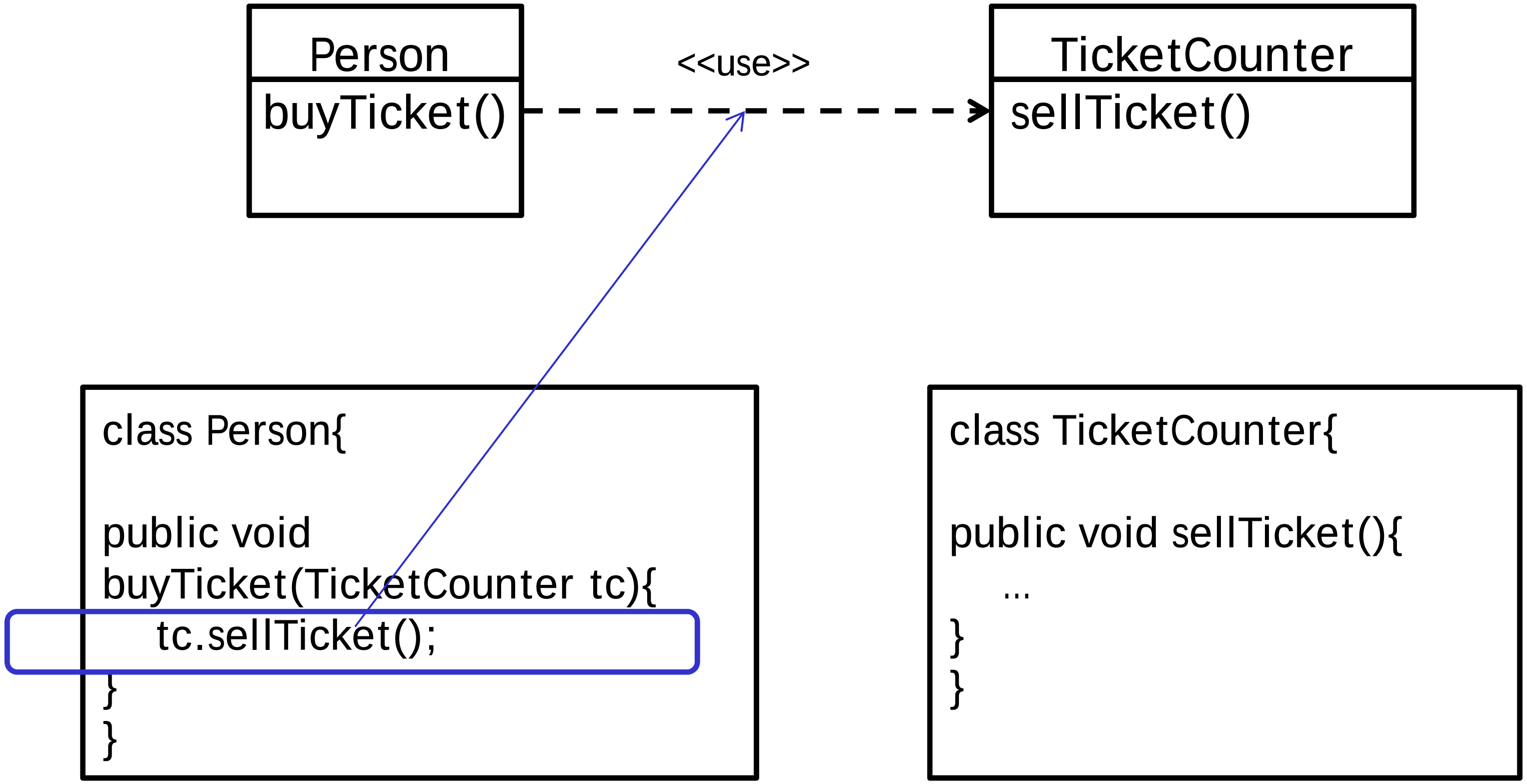
Interface, however, does not have the benefits of code reuse; D1 and D2 have to implement the abstract operation code().

```
public class D1 extends A {
  public int code() {
  message();   return 1;}
}
public class D2 extends A {
  public int code() { return 2;}
}
```

```
public class D1 implements A {
  public int code() { return 10;}
}

public class D2 implements A {
  public int code() { return 20;}
}
```

Dependency

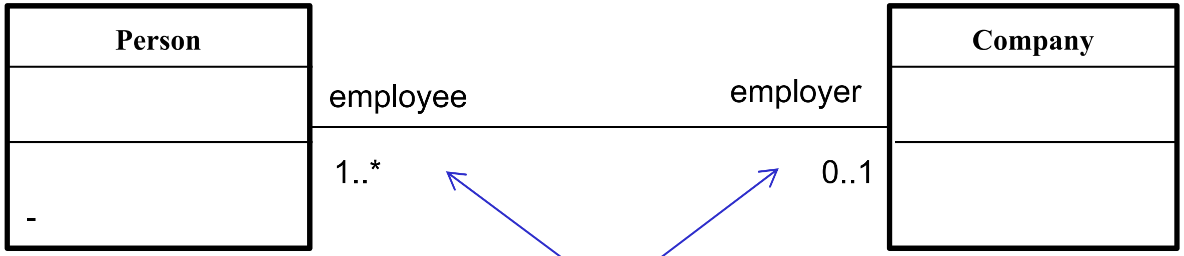
- A dependency states that the implementation or functioning of one or more elements requires the presence of one or more other elements.



```
class Person{

public void
buyTicket(TicketCounter tc){
    tc.sellTicket();
}
}
```

```
class TicketCounter{

public void sellTicket(){
    ...
}
}
```

Association Example

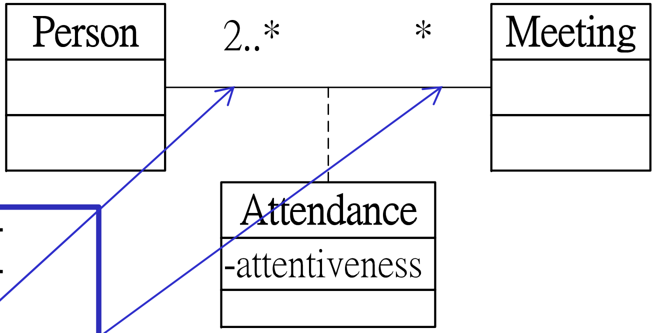


```
public class Person {
  private  Company employer;

  public void setEmployer (Company c){
              employer = c;
  }
  public Company getEmployer(){
              return employer;
   }
}
```

```
public class Company {
  private  Set<Person> employee;

  public void addEmployee(Person p){
            employee.add(p);
   }
  public Set<Person> getEmployee(){
            return employee;
   }
}
```

Association Class

- ❑ An association has attributes associated with the association itself (not just the participating objects)
- ❑ Implementation
  - ➢ Each participating object contains a reference to the association class object
  - ➢ The association class object contains references to each of the related objects



```
class Person {
  …
  (some sort of collection of references
to Attendance objects) attendance;
  …
}
```

```
class Meeting {
  …
  (some sort of collection of references
to Attendance objects) attendance;
  …
}
```

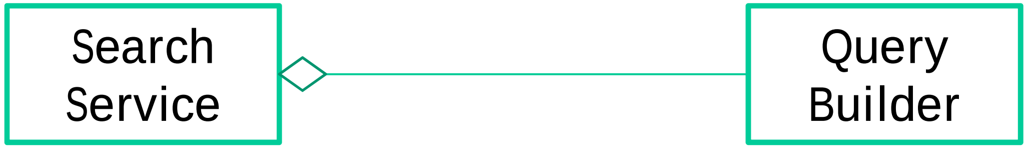
```
class Attendance {
  …
  Person person;
  Meeting meeting;
  int attentiveness;
  …
}
```

Aggregation Type

- An association may represent a composite aggregation (i.e., composition or a whole/part relationship).
  - Composite aggregation is a strong form of aggregation that requires a part instance be included in **at most one composite** at a time. (*Composition*)
  - If a composite is deleted, all of its parts are normally deleted with it.
- Aggregation type could be:
  - Shared aggregation (aggregation)
  - Composite aggregation (composition)

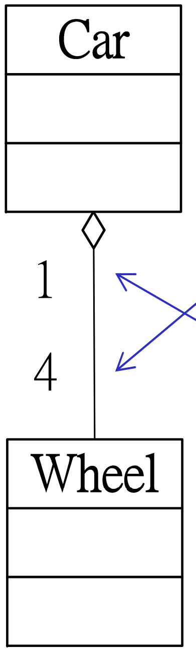
Aggregation

- ❑ Aggregation is a “weak” form of aggregation when part instance is independent of the composite:
  - ➢ The same (shared) part could be included in several composites, and
  - ➢ If composite is delete, shard parts may still exist.



Search Service has a **Query Builder** using shared aggregation

Aggregation Example



```
public class Car {
  private Wheel wheel1, wheel2, wheel3, wheel4;
  …
}
```

```
public class Wheel {
  private Car car;
  …
}
```

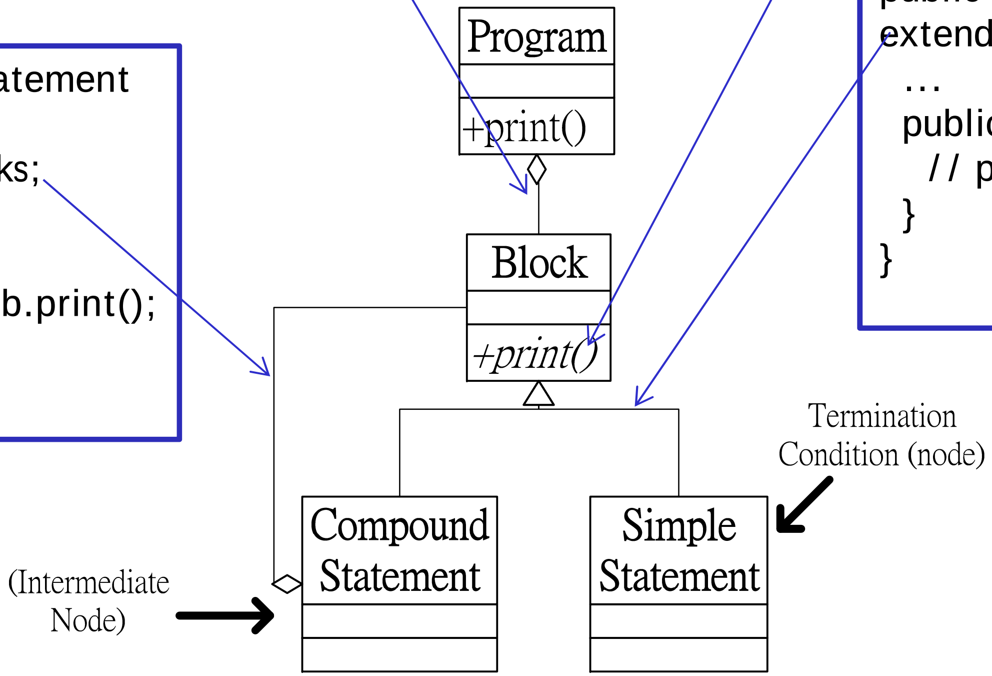
Recursive Aggregation

```
public class Program {
  private Set<Block> blocks;
  ...
  public void print() {
    for (Block b in blocks)
          b.print();
  }
}
```

```
public class Block {
  ...
  public abstract void print();
}
```

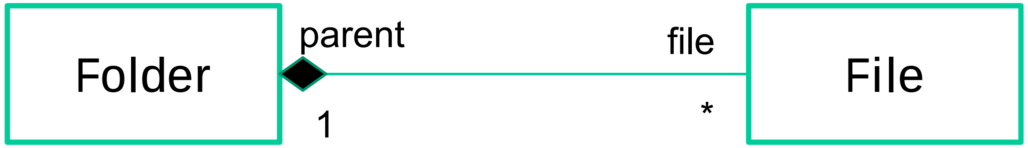
```
public class CompoundStatement
extends Block {
  private Set<Block> blocks;
  ...
  public void print() {
    for (Block b in blocks) b.print();
  }
}
```

```
public class SimpleStatement
extends Block {
  ...
  public void print() {
    // print statement
  }
}
```



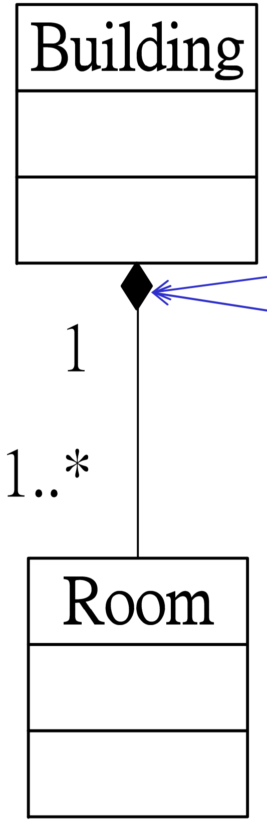
Composition

- ❑ Composition is a “strong” form of aggregation where the whole and parts have coincident lifetimes.
  - ➢ It is a whole/part relationship,
  - ➢ It is binary association,
  - ➢ Part could be included in at most one composite (whole) at a time,
  - ➢ If a composite (whole) is deleted, all of its composite parts are “normally” deleted with it.
- ❑ A **Composition** adds a lifetime responsibility to *Aggregation*



Folder could contain many **files**, while each **File** has exactly one **Folder** parent. If **Folder** is deleted, all contained Files are deleted as well.

Composition Example



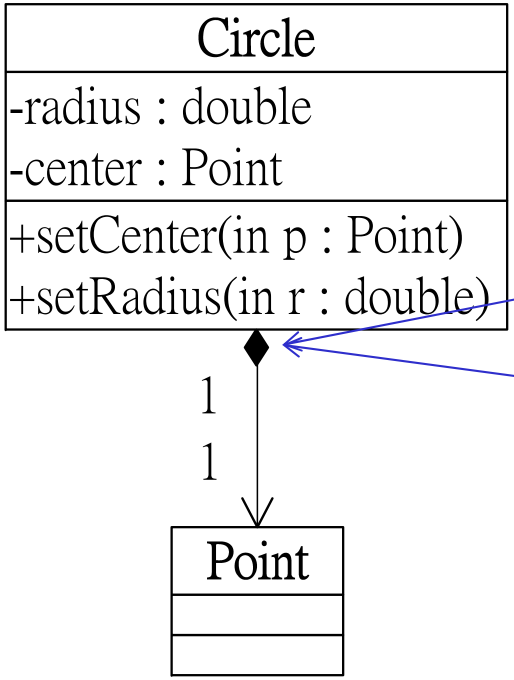
```
public class Building {
  private Set<Room> rooms;

  public Building() {
    rooms = new Set<Room>();
  }

  protected void finalize() {
    rooms = null;
  }
}
```

Create Room objects

Destroy Room objects

```
public class Room {
  private …;
  private …;
  …
}
```

Composition Example



```
public class Circle {
  double radius;
  Point center;

  public Circle(Point c, double r) {
    this.radius = r;
    this.center = c;
}

  protected void finalize() {
    this.radius = 0;
    this.center = null;
  }
  ...
}
```

Create Point object

Destroy Point object

```
public class Point {
  private …;
  private …;
  …
}
```

Lab 1

- ❑ A country has a capital city.
- ❑ A dining philosopher is using a fork.
- ❑ A file is an ordinary file or a directory file.
- ❑ Files contain records.
- ❑ A polygon is composed of an ordered set of points.
- ❑ A drawing object is text, a geometrical object, or a group.

Homework

- Model the following problem statement based on UML class diagram
  - A person has a name, address, and social security number. A person may charge time to projects and earn a salary. A company has a name, address, phone number, and primary product. A company hires and fires persons. Person and Company have a many-to-many relationship.
  - There are two types of persons: workers and managers. Each worker works on many projects; each manager is responsible for many projects. A project is staffed by many workers and exactly one manager. Each project has a name, budget, and internal priority for securing resources.
  - A company is composed of multiple departments; each department within a company is uniquely identified by its name. A department usually, but not always, has a manager. Most managers manage a department; a few managers are not assigned to any department. Each department manufactures many products; while each product is made by exactly one department. A product has a name, cost, and weight.