برنامه نویسی پیشرفته
C#

۱۷ آذر ۹۸

ملکی مجد

topics

- A brief introduction to UML
- Write enumerator for your collection

UML = Unified Modeling Language

- Class diagram, object diagram, sequence diagram and package diagram

- a **class diagram** in UML
  - a type of static structure diagram that describes the structure of a system by showing the **system's classes**, **their attributes**, **operations** (or methods), and the relationships among objects.
  - the main building block of object oriented modeling

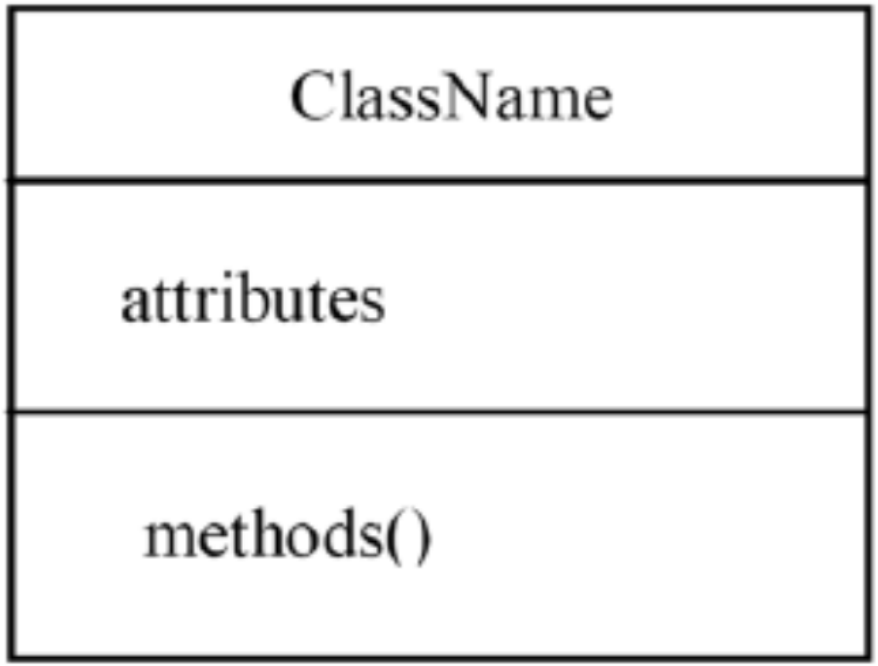
Classes in diagram

- classes are represented with boxes that contain three compartments:

1. The top compartment contains the **name of the class**. It is printed in bold and centered, and the first letter is capitalized.
2. The middle compartment contains the **attributes of the class**. They are left-aligned and the first letter is lowercase.
3. The bottom compartment contains the **operations** the class can execute. They are also left-aligned and the first letter is lowercase.

- Note: To further describe the behavior of systems, the class diagrams can be complemented by a state diagram.

A class



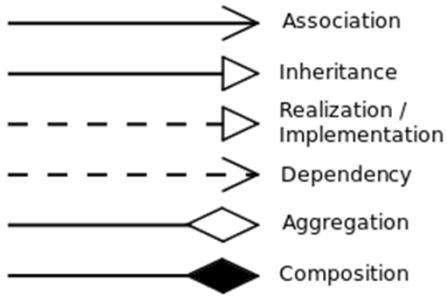
- specify the visibility of a class member
  - + public
  - - private
  - # protected

- Scope
  - Classifier members (static)
    - To indicate a classifier scope for a member, its name must be underlined.
  - Instance members

Relationships

- Class-level relationships
- Instance-level relationships
- General relationship

UML relations notation



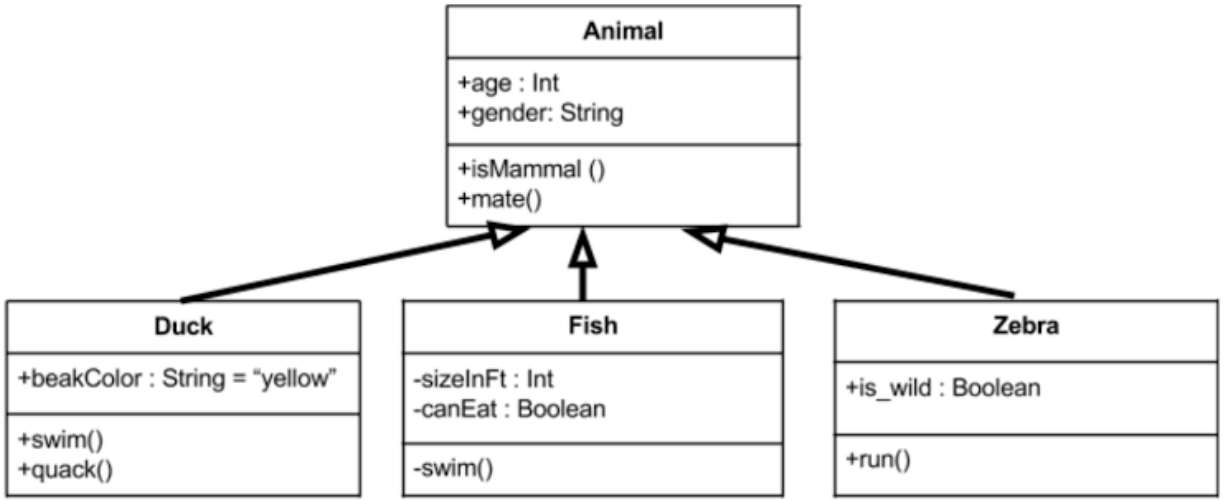
Class-level relationships

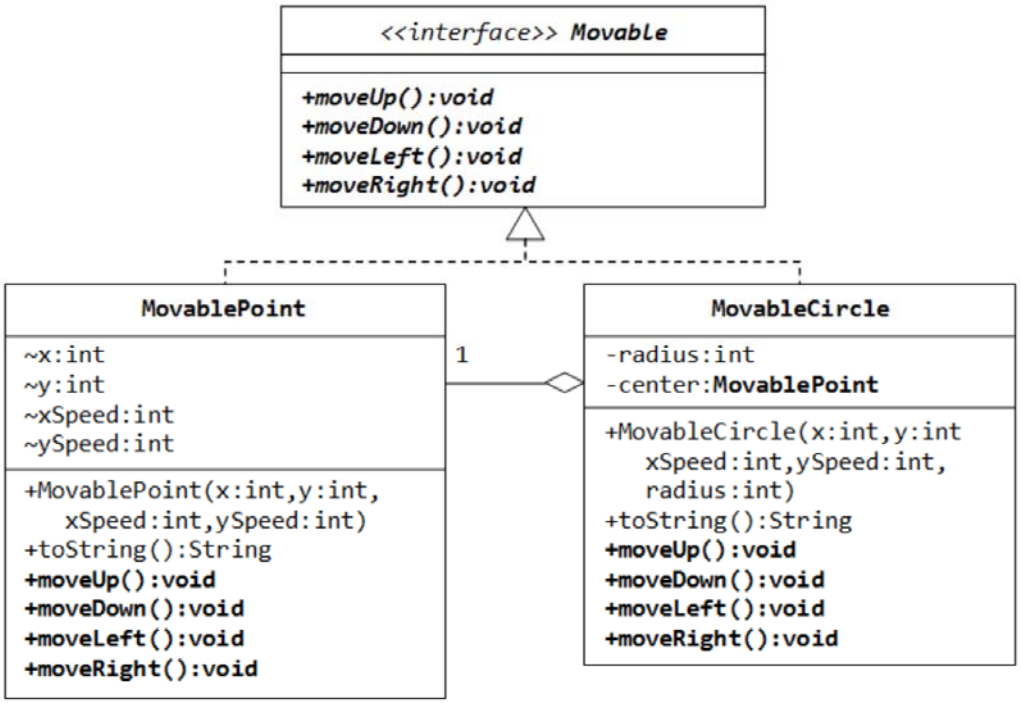
- **Generalization/Inheritance**
  - known as the *inheritance* or *"is a"* relationship
  - The UML graphical representation of a Generalization is a hollow triangle shape on the superclass end of the line (or tree of lines) that connects it to one or more subtypes.
- **Realization/Implementation**
  - The UML graphical representation of a Realization is a hollow triangle shape on the interface end of the *dashed* line (or tree of lines) that connects it to one or more implementers.
  - A realization is a relationship between classes, interfaces, components and packages that connects a client element with a supplier element.
  - A realization relationship between classes/components and interfaces shows that the class/component realizes the operations offered by the interface.

Instance-level relationships

- **Dependency**
  - a semantic connection between dependent and independent model elements
  - it exists between two elements if changes to the definition of one element may cause changes to the other.
- **Association**
  - a relationship between two separate classes.
- **Aggregation**
  - a special form of association which is a unidirectional
  - “has a” or “is part of” relationship
- **Composition**
  - a restricted form of Aggregation in which two entities (or you can say classes) are highly dependent on each other. (human and heart)

example


Animal
+age : Int
+gender: String
+isMammal ()
+mate()
Duck
+beakColor : String = “yellow”
+swim()
+quack()
Fish
-sizeInFt : Int
-canEat : Boolean
-swim()
Zebra
+is_wild : Boolean
+run()


<<interface>> Movable
+moveUp():void
+moveDown():void
+moveLeft():void
+moveRight():void
MovablePoint
~x:int
~y:int
~xSpeed:int
~ySpeed:int
+MovablePoint(x:int,y:int,
xSpeed:int,ySpeed:int)
+toString():String
+moveUp():void
+moveDown():void
+moveLeft():void
+moveRight():void
1
MovableCircle
-radius:int
-center:MovablePoint
+MovableCircle(x:int,y:int
xSpeed:int,ySpeed:int,
radius:int)
+toString():String
+moveUp():void
+moveDown():void
+moveLeft():void
+moveRight():void

- How define an enumerator that you can use to iterate over the elements in a collection.

Enumerating the elements in array

```
int[] pins = { 9, 3, 7, 2 };
        foreach (int pin in pins)
        {
            Console.WriteLine(pin);
        }
```

an enumerable collection

- a collection that implements the *System.Collections.IEnumerable* interface.

```
public interface IEnumerable
{
        IEnumerator GetEnumerator();
}
public interface Enumerator
{
        object Current { get; }
        bool MoveNext();
        void Reset();
}
```

Think of an enumerator as a pointer indicating elements in a list.

Add enumeration to class tree<Item>

A simple iterator

```
using System;using System.Collections.Generic;using System.Collections;
class BasicCollection<T> : IEnumerable<T>{
        private List<T> data = new List<T>();
        public void FillList(params T [] items){
                foreach (var datum in items){
                        data.Add(datum);      }
        }
        IEnumerator<T> IEnumerable<T>.GetEnumerator(){
                foreach (var datum in data){
                        yield return datum;}
}
IEnumerator IEnumerable.GetEnumerator(){ // Not implemented in this example
throw new NotImplementedException();}
```

alternative iteration mechanisms

```
class BasicCollection<T> : IEnumerable<T>{
        ...
        public IEnumerable<T> Reverse        {
                get
                {
                        for (int i = data.Count - 1; i >= 0; i--){
                                yield return data[i];
                        }
                }
        }
}
```

```
BasicCollection<string> bc = new BasicCollection<string>();
bc.FillList("Twas", "brillig", "and", "the", "slithy", "toves");

foreach (string word in bc){
        Console.WriteLine(word);
        }
foreach (string word in bc.Reverse){
        Console.WriteLine(word);
        }
```