جامعة الأميرة نورة بنت عبد الرحمن
Princess Nora Bint Abdul Rahman University

CS313D: ADVANCED PROGRAMMING LANGUAGE

Computer Science department

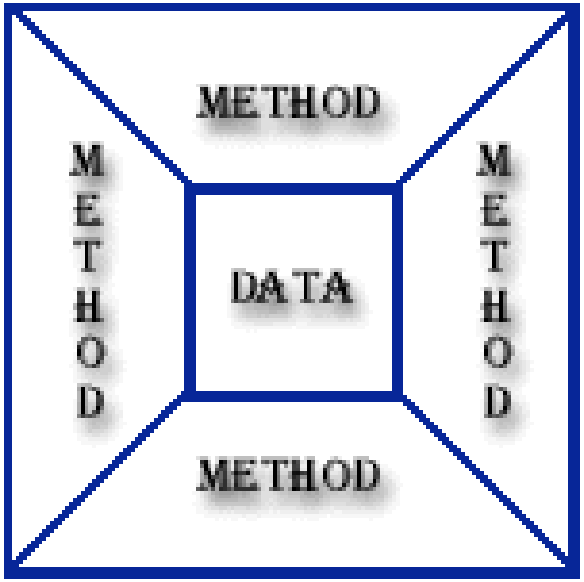
Lecture 4 (a): Classes & Objects

Lecture Contents

- What is a class?
- Class definition:
  - Data
  - Methods
  - Constructors
  - objects
- Static members
- Readonly members
- composition

What is a class?

- Class:
  - like a *struct* in C++, defines a data type (blueprint)
  - Combines data and operations in one place:
    - data / fields/ properties / state variables
    - Operations/ behavior / methods that modify state

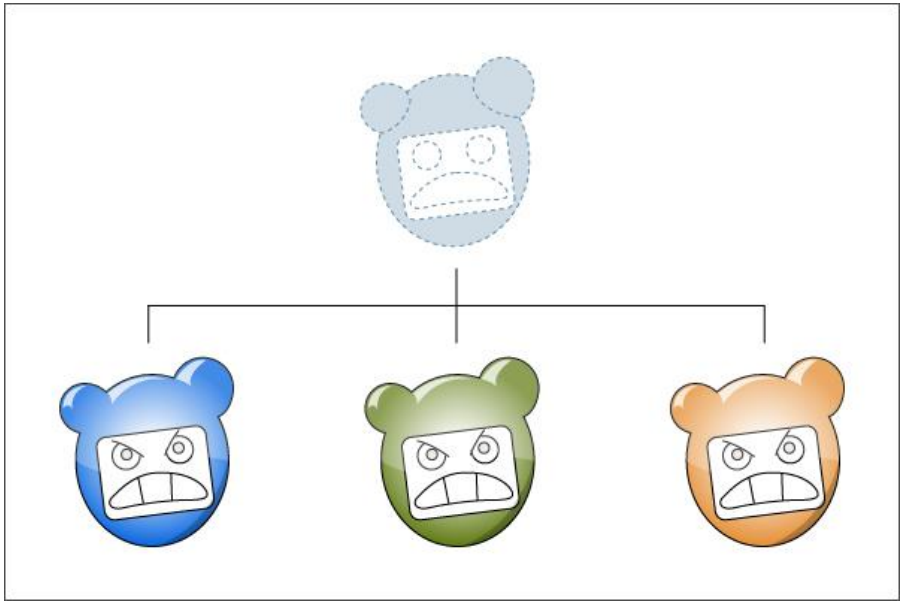


4 Encapsulation

- The bundling of data and procedures(methods) into a single unit(called class).
- Class : various data elements and member functions are wrapped up together.
- main feature of object oriented programming

Objects & Classes

- variables of class type are objects
- Object:
  - an instance / realization of a class
  - same as relationship between a *dataType* and its variables



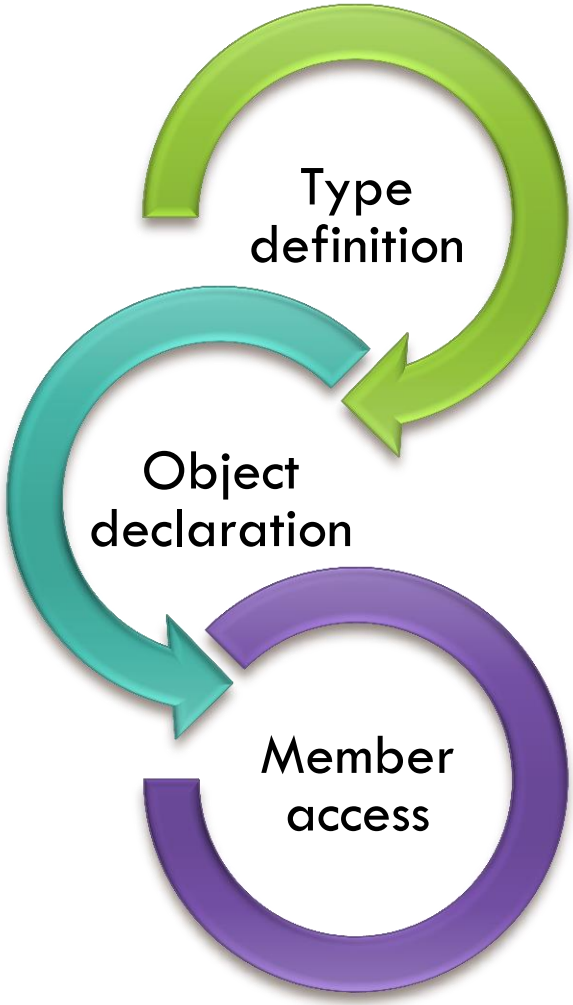
Blueprint of a house

Design → build → use

Attributes??

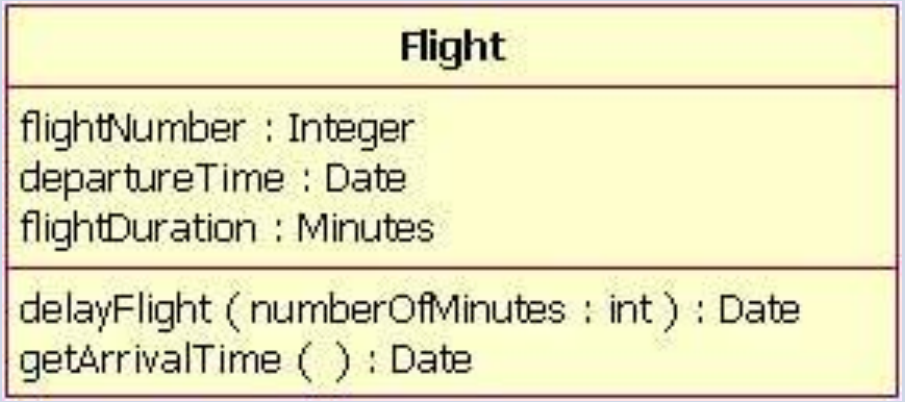
Methods??

How to define a Class??



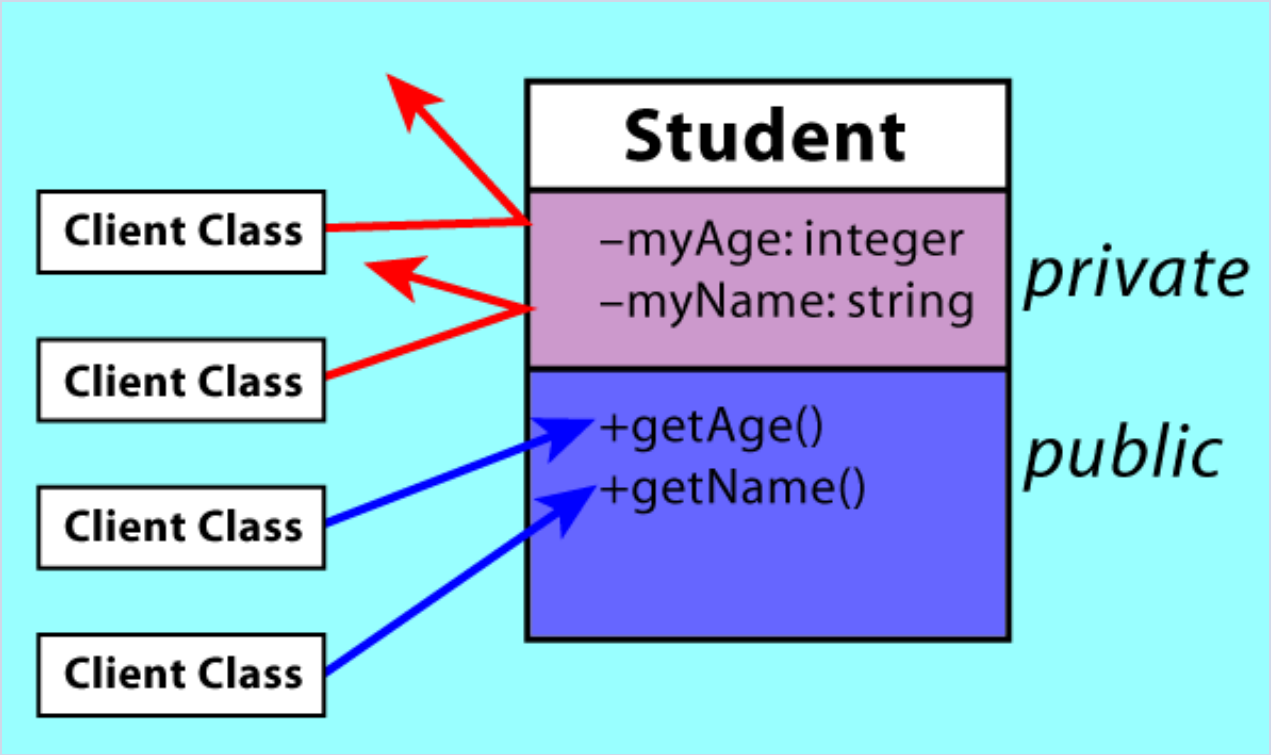
the Unified Modeling Language (UML)

- can be defined as a modeling language to capture the architectural, behavioral and structural aspects of a system.
- The UML has an important role in object oriented analysis and design.
- Objects are the key to this object oriented world → The basic requirement of is to identify the object efficiently.



Examples

What about a student??



Access modifiers

- Public items (usually member functions) are "user-accessible“
- Outside of class definition, cannot change (or even access) private data

Mark	Visibility type
+	Public
#	Protected
-	Private
~	Package

Marks for UML-supported visibility types

Initializing data with Constructors

- A constructor can be used to initialize an object of a class when the object is created.
- Every class must have at least one constructor.
  - If you do not provide any constructors in a class's declaration → the compiler creates a default constructor that takes no arguments when it is invoked.
  - if you declare any constructors → no default constructor will be created for that class
  - With the default constructor, its instance variables are initialized to their default values.

Common Programming Error 10.3

If a class has constructors, but none of the `public` constructors are parameterless constructors, and an attempt is made to call a parameterless constructor to initialize an object of the class, a compilation error occurs. A constructor can be called with no arguments only if the class does not have any constructors (in which case the default constructor is called) or if the class has a parameterless constructor.

Be careful!!

Overloaded Constructors

- Overloaded constructors
  - enable objects of a class to be initialized in different ways.
  - multiple constructor declarations with different signatures.
  - the compiler differentiates signatures by :
    - the *number* of parameters,
    - the *types* of the parameters and
    - the *order* of the parameter types in each signature.

Time Class Case Study

- Class `Time1` represents the time of day in universal-time format (24-hour clock format) .
  - instance variables `hour`, `minute` and `second`
- methods:
  - `Property set/get`
  - `five overloaded Constructors`
  - `setTime`
  - `ToUniversalString` (hh:mm:ss)
  - `ToString (hh:mm:ss AM/PM)`

The Time2 Class

Private data

Overloaded constructors

Constructor initializers → Using this operator

```
1  // Fig. 10.5: Time2.cs
2  // Time2 class declaration with overloaded constructors.
3  using System; // for class ArgumentOutOfRangeException
4
5  public class Time2
6  {
7     private int hour; // 0 - 23
8     private int minute; // 0 - 59
9     private int second; // 0 - 59
10
11    // constructor can be called with zero, one, two or three arguments
12    public Time2( int h = 0, int m = 0, int s = 0 )
13    {
14       SetTime( h, m, s ); // invoke SetTime to validate time
15    } // end Time2 three-argument constructor
16
17    // Time2 constructor: another Time2 object supplied as an argument
18    public Time2( Time2 time )
19       : this( time.Hour, time.Minute, time.Second ) { }
20
```

Fig. 10.5 | Time2 class declaration with overloaded constructors. (Part 1 of 5.)

The Time2 Class

get/set

```
21      // set a new time value using universal time; ensure that
22      // the data remains consistent by setting invalid values to zero
23      public void SetTime( int h, int m, int s )
24      {
25         Hour = h; // set the Hour property
26         Minute = m; // set the Minute property
27         Second = s; // set the Second property
28      } // end method SetTime
29
30      // property that gets and sets the hour
31      public int Hour
32      {
33         get
34         {
35            return hour;
36         } // end get
37         set
38         {
39            if ( value >= 0 && value < 24 )
40               hour = value;
41            else
42               throw new ArgumentOutOfRangeException(
43                  "Hour", value, "Hour must be 0-23" );
44         } // end set
45      } // end property Hour
```

Fig. 10.5 | Time2 class declaration with overloaded constructors. (Part 2 of 5.)

The Time2 Class

get/set

```
46
47      // property that gets and sets the minute
48      public int Minute
49      {
50         get
51         {
52            return minute;
53         } // end get
54         set
55         {
56            if ( value >= 0 && value < 60 )
57               minute = value;
58            else
59               throw new ArgumentOutOfRangeException(
60                  "Minute", value, "Minute must be 0-59" );
61         } // end set
62      } // end property Minute
63
```

Fig. 10.5 | Time2 class declaration with overloaded constructors. (Part 3 of 5.)

The Time2 Class

Convert to string representation

```
64      // property that gets and sets the second
65      public int Second
66      {
67         get
68         {
69            return second;
70         } // end get
71         set
72         {
73            if ( value >= 0 && value < 60 )
74               second = value;
75            else
76               throw new ArgumentOutOfRangeException(
77                  "Second", value, "Second must be 0-59" );
78         } // end set
79      } // end property Second
80
81      // convert to string in universal-time format (HH:MM:SS)
82      public string ToUniversalString()
83      {
84         return string.Format(
85            "{0:D2}:{1:D2}:{2:D2}", Hour, Minute, Second );
86      } // end method ToUniversalString
87
```

Fig. 10.5 | Time2 class declaration with overloaded constructors. (Part 4 of 5.)

The Time2 Class

Convert to string representation

```
88      // convert to string in standard-time format (H:MM:SS AM or PM)
89      public override string ToString()
90      {
91         return string.Format( "{0}:{1:D2}:{2:D2} {3}",
92            ( ( Hour == 0 || Hour == 12 ) ? 12 : Hour % 12 ),
93            Minute, Second, ( Hour < 12 ? "AM" : "PM" ) );
94      } // end method ToString
95   } // end class Time2
```

Fig. 10.5 | Time2 class declaration with overloaded constructors. (Part 5 of 5.)

Test!!


```
1  // Fig. 10.6: Time2Test.cs
2  // Overloaded constructors used to initialize Time2 objects.
3  using System;
4
5  public class Time2Test
6  {
7     public static void Main( string[] args )
8     {
9        Time2 t1 = new Time2(); // 00:00:00
10       Time2 t2 = new Time2( 2 ); // 02:00:00
11       Time2 t3 = new Time2( 21, 34 ); // 21:34:00
12       Time2 t4 = new Time2( 12, 25, 42 ); // 12:25:42
13       Time2 t5 = new Time2( t4 ); // 12:25:42
14       Time2 t6; // initialized later in the program
15
16       Console.WriteLine( "Constructed with:\n" );
17       Console.WriteLine( "t1: all arguments defaulted" );
18       Console.WriteLine( "   {0}", t1.ToUniversalString() ); // 00:00:00
19       Console.WriteLine( "   {0}\n", t1.ToString() ); // 12:00:00 AM
20
```

Fig. 10.6 | Overloaded constructors used to initialize `Time2` objects. (Part 1 of 5.)

Test


```
21          Console.WriteLine(
22             "t2: hour specified; minute and second defaulted" );
23          Console.WriteLine( "   {0}", t2.ToUniversalString() ); // 02:00:00
24          Console.WriteLine( "   {0}\n", t2.ToString() ); // 2:00:00 AM
25
26          Console.WriteLine(
27             "t3: hour and minute specified; second defaulted" );
28          Console.WriteLine( "   {0}", t3.ToUniversalString() ); // 21:34:00
29          Console.WriteLine( "   {0}\n", t3.ToString() ); // 9:34:00 PM
30
31          Console.WriteLine( "t4: hour, minute and second specified" );
32          Console.WriteLine( "   {0}", t4.ToUniversalString() ); // 12:25:42
33          Console.WriteLine( "   {0}\n", t4.ToString() ); // 12:25:42 PM
34
35          Console.WriteLine( "t5: Time2 object t4 specified" );
36          Console.WriteLine( "   {0}", t5.ToUniversalString() ); // 12:25:42
37          Console.WriteLine( "   {0}", t5.ToString() ); // 12:25:42 PM
38
```

Fig. 10.6 | Overloaded constructors used to initialize `Time2` objects. (Part 2 of 5.)

Test


```
39          // attempt to initialize t6 with invalid values
40          try
41          {
42             t6 = new Time2( 27, 74, 99 ); // invalid values
43          } // end try
44          catch ( ArgumentOutOfRangeException ex )
45          {
46             Console.WriteLine( "\nException while initializing t6:" );
47             Console.WriteLine( ex.Message );
48          } // end catch
49       } // end Main
50    } // end class Time2Test
```

Fig. 10.6 | Overloaded constructors used to initialize `Time2` objects. (Part 3 of 5.)

```
Constructed with:

t1: all arguments defaulted
   00:00:00
   12:00:00 AM

t2: hour specified; minute and second defaulted
   02:00:00
   2:00:00 AM

t3: hour and minute specified; second defaulted
   21:34:00
   9:34:00 PM

t4: hour, minute and second specified
   12:25:42
   12:25:42 PM

t5: Time2 object t4 specified
   12:25:42
   12:25:42 PM
```

Fig. 10.6 | Overloaded constructors used to initialize `Time2` objects. (Part 4 of 5.)

```
Exception while initializing t6:
hour must be 0-23
Parameter name: hour
Actual value was 27.
```

Fig. 10.6 | Overloaded constructors used to initialize `Time2` objects. (Part 5 of 5.)

The destructor

- Object *destructor*:
  - Same as the class name, preceded by a tilde, with no access modifier in its header.
  - called by the Common Language Runtime (CLR) garbage collector to perform termination housekeeping on an before its memory is reclaimed.
  - not guaranteed to execute at a specified time.
  - Rarely used

`static` Class Members

- `static` field:
  - called a class variable
  - represents class-wide information—only one copy is be shared by all objects of a class.
  - The declaration begins with the keyword `static`.
  - `public static` members can be accessed through a reference to any object of the class, or by qualifying the member name with the class name and a dot (`.`).
  - A class's `private static` class members can be accessed only through the methods and properties of the class.
    - To access a private static member from outside its class, a public static method or property can be provided

Software Engineering Observation 10.8

Static variables, methods and properties exist, and can be used, even if no objects of that class have been instantiated.

Tip !!

readonly Instance Variables

- C# provides keyword readonly to specify that an instance variable of an object is not modifiable and that any attempt to modify it after the object is constructed is an error.
  - Like constants, readonly variables are declared with all capital letters by convention
  - readonly instance variables can be initialized when they are declared, but this is not required.
  - Members that are declared as const must be assigned values at compile time, whereas members declared with keyword readonly, can be initialized at execution time.
  - If a class provides multiple constructors, every constructor should initialize a readonly variable.

Composition

- A class can have references to objects of other classes as members.
- This is called composition and is sometimes referred to as a has-a relationship.
- Examples:
  - An `AlarmClock` object needs two references to `Time` objects to know the current time and the time when it's supposed to sound its alarm.
  - A `Robot` Object has-a `MechanicalArm`
  - A `car` Object has-a `Wheel`
  - A `student` Object has-a `BirthDate`
  - `You name it !!!`

That's all for today

Chapter 10