



Object Oriented Programming

Lecture 4 Constructor

Arsalan Rahman Mirza

Computer Science Department

Faculty of Science

2022-2023

Class Scope (I)

- A class's instance variables and methods belong to that class's *scope*.
- Within a class's scope, *class* members are immediately accessible to all of that class's methods and can be referenced by name.
- **Outside** a class's scope, class members cannot be referenced directly by name.
- Those class members that are visible (such as public members) can be accessed only through a "handle".
 - members can be referenced via the format *referenceName.memberName*

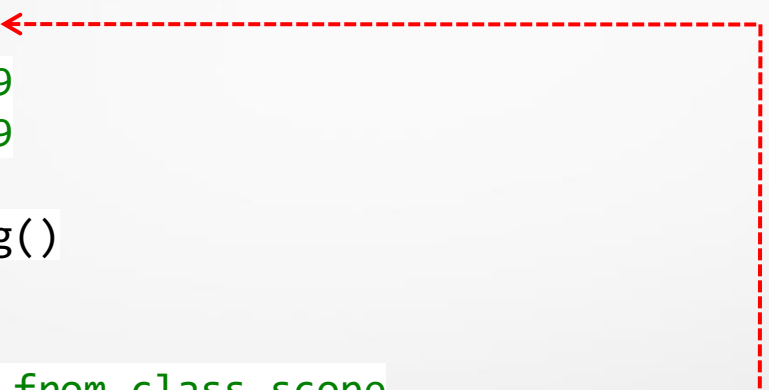
Class Scope (II)

- If a variable is defined in a method, only that method can access the variable
 - i.e., the variable is a local variable of that method.
 - Called *block scope*
- If a method defines a variable that has the same name as a variable with class scope, the method-scope variable hides the class-scope variable in that method's scope.
- A hidden instance variable can be accessed in a method by preceding its name with the keyword `this` and the dot operator, as in `this.hour`.

Class Scope (III)

```
class Time1
{
    private int hour; // 0-23
    private int minute; // 0-59
    private int second; // 0-59

    public string ToUniversalString()
    {
        int hour;
        this.hour = 5; // hour from class scope
        return String.Format("{0:D2}:{1:D2}:{2:D2}", this.hour, minute, second);
    }
}
```



Constructor (I)

- A **constructor** is a member method that is used to initialize data members of a class
- Is called automatically when an object of the class is created using the `new` operator
- Must be a `public` member method
- Must be named the same as the class
- Must have no return type

Constructor (II)

- Instance variables can be initialized either by a constructor or when they are declared in the class body.
- A class can contain overloaded constructors to provide multiple ways to initialize objects of that class
- Regardless of whether instance variables receive explicit initialization values, the instance variables always are initialized.
- In such cases, instance variables receive their default values.
 - 0 for primitive numeric type variables
 - false for bool variable
 - null for references



Performance Tip

Because instance variables always are initialized to default values by the runtime, avoid initializing instance variables to their default values in the constructor.

Constructor (III)

- When creating an object of a class, the programmer can provide *initializers in parentheses* to the right of the class name.
- These initializers are the arguments to the constructor.
- The general form
 - *ClassName* *objectReference* = **new** *ClassName*(*arguments*);
 - **new** indicates that an object is being created.
 - *ClassName* indicates the type of the new object.
 - *arguments* specifies a comma-separated list of the values used by the constructor to initialize the object
- See the example

Different types of constructors

Default constructor
of the class

class Time1

```
{
    private int hour; // 0-23
    private int minute; // 0-59
    private int second; // 0-59
    public void setTime(int hourValue, int minuteValue, int secondValue)
    {
        hour = (hourValue >= 0 && hourValue < 24) ? hourValue : 0;
        minute = (minuteValue >= 0 && minuteValue < 60) ? minuteValue : 0;
        second = (secondValue >= 0 && secondValue < 60) ? secondValue : 0;
    }
    public string ToUniversalString()
    {
        return String.Format("{0:D2}:{1:D2}:{2:D2}", hour, minute, second);
    }
    public string ToStandardString()
    {
        return String.Format("{0}:{1:D2}:{2:D2} {3}",
            ((hour == 12 || hour == 0) ? 12 : hour % 12), minute,
            second, (hour < 12 ? "AM" : "PM"));
    }
    // Time1 constructor initializes instance variables to
    // zero to set default time to midnight
}
```

public Time1()

```
{
    setTime(0, 0, 0);
}
// Time2 constructor: hour supplied, minute and second
// defaulted to 0
public Time1(int hour)
{
    setTime(hour, 0, 0);
}
// Time2 constructor: hour and minute supplied, second
// defaulted to 0
public Time1(int hour, int minute)
{
    setTime(hour, minute, 0);
}
// Time2 constructor: hour, minute and second supplied
public Time1(int hour, int minute, int second)
{
    setTime(hour, minute, second);
}
// Time2 constructor: initialize using another Time2 object
public Time1(Time1 time)
{
    setTime(time.hour, time.minute, time.second);
}
}
```

Constructor (IV)

- Like methods, constructors of a class can be overloaded.
- The previous example overloads the constructor to provide a variety of ways to initialize **Time1 objects**.
- Each constructor calls Time1 method SetTime, which ensures that the object begins in a consistent state by setting out of-range values to zero.
- C# invokes the appropriate constructor by matching the number, types and order of the arguments

Constructor (V)

- The Default Constructor
 - Constructors can have any number of parameters, including none
 - A **default constructor** is one that takes no arguments either due to
 - No parameters or
 - All parameters have default values
 - If a class has any programmer-defined constructors, it must have a programmer- defined default constructor

Common Programming Error

If a class has constructors, but none of the public constructors is a default constructor, and a program attempts to call a no-argument constructor to initialize an object of the class, a compilation error occurs. A constructor can be called with no arguments only if there are no constructors for the class (in which case the compiler-provided default constructor is called) or if the class defines a public no-argument constructor.

Default Constructor

- If no constructor is defined, the default constructor is being called

```
class Book
{
    public string title;
    public string ISSN;
    public int pages;
    public string data;
}
static void Main(string[] args)
{
    Book any = new Book(); // default constructor
}
```

Constructors with Parameters

- A constructor, that is being used to set values of members, usually defined as public.

```
class Book
{
    public int pages;
    public string data;
    public Book(int pg, string dt)
    {
        pages = pg;
        data = dt;
    }
    static void Main(string[] args)
    {
        Book any = new Book(5, "Ahmad");
        //calling constructor
    }
}
```

Constructors with Parameters

- NOTE: if a class has a constructor with parameters, the default constructor will not work.

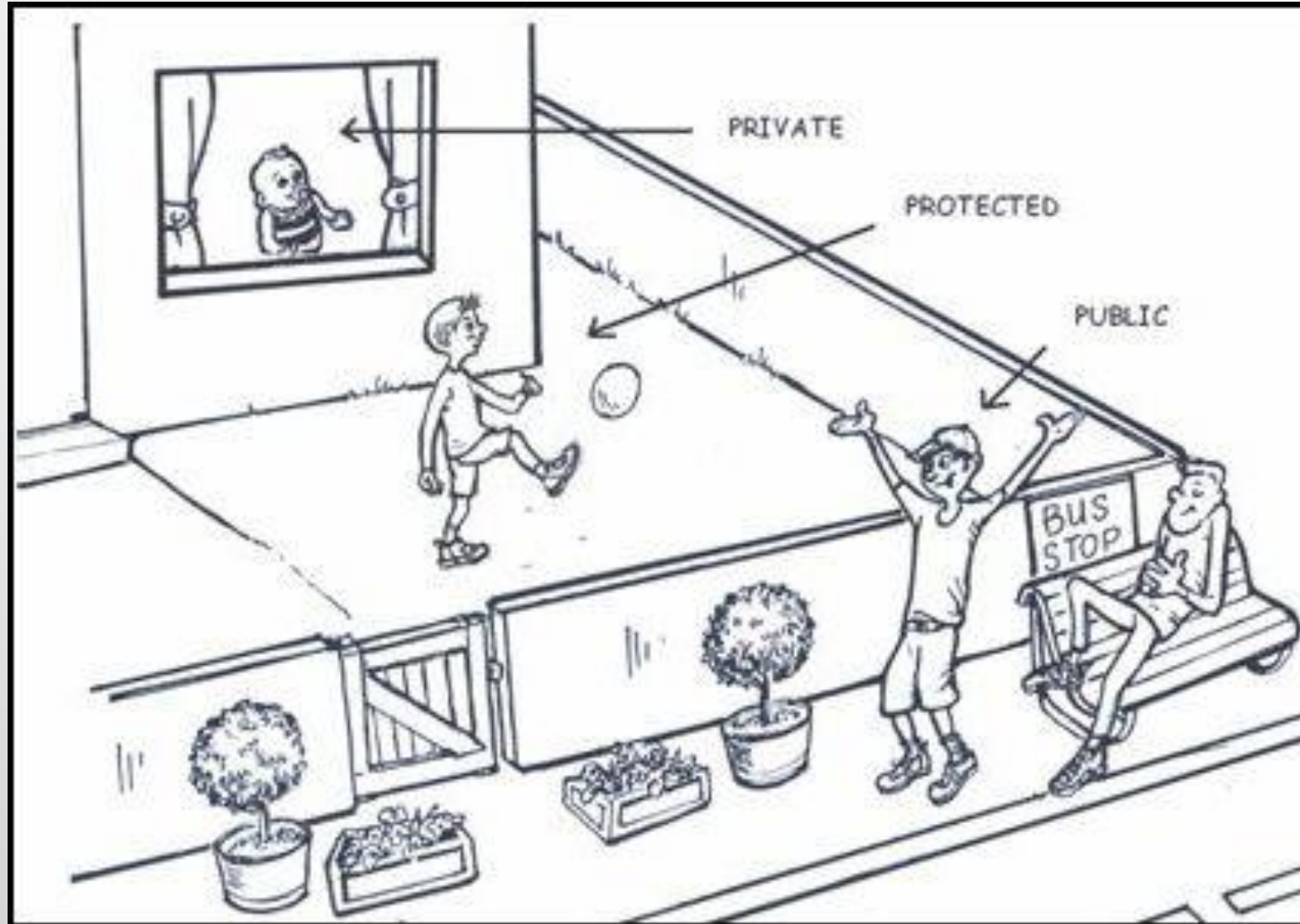
```
class Book
{
    public Book(string book_title, int n_authors)
    {
    }
    // the rest of book class ...
}
static void Main(string[] args)
{
    Book any = new Book();           // ERROR
}
```

Access Modifiers

There are 3 main types of Access Modifiers:

- **Private:** The type or member can be accessed only by code in the same class.
- **Protected:** The type or member can be accessed only by code in the same class, or in a class that is derived from that class.
- **Public:** The type or member can be accessed by any other code.

Access Modifiers



A simple class example-default constructor will be created automatically

```
class book
{
    public string title;
    public string ISSN;
    private int pages;
    private string data;
}
class Author
{
    public string name;
    ushort age;
    string email;
}
```

```
class Program
{
    static void Main(string[] args)
    {
        book b = new book();
        b.title = "my book";
        b.ISSN = "445X_4480";

        Author a = new Author();
        a.name = "Kawe";
        a.email = "Kawe123@gmail.com";
        Console.WriteLine(a.name);
        Console.ReadKey();
    }
}
```

// by default , all variables in a class are private.