W1:

High level statement:

If top-level is used, the compiler will create the required class and the Main method itself automatically.

C# is object-oriented.

E.g., functions belong to some types such as class. A function is a member of a class called method.

You cannot have a variable outside a class (called a field) or a method (a local variable either.

Namespaces are used in C# to organize and provide a level of separation of codes. They can be considered as a container which consists of other namespaces, classes, etc. E.g., namespace. nested\_namespave.class.method();

.Net:

Compiler of a .Net language (like C#, called managed code) takes a source code file, and produces an intermediate output file called an assembly. The code in an assembly is not native machine code, but an intermediate language (called CIL)

Only when the program is called to run, a CLR just-in-time compiler compiles it to the native machine code.

Graphical user interface, text

Description automatically generated with medium confidence

Week 2:

A type is a set of values and the operations that permitted on the values. All simple pre-defined types are value types. The non-simple predefined types of string and object are reference types.

A class is a type that stores logically related data and functions

A class contains data members and function members.

E.g.,

Class Student

{

string name;

int yearLevel; //field where we store the data

}

Properties: private (memory allocated) public (no memory allocated)

E.g.,

int Property\_name

{

Set

{

//write the code to set the value

}

Get

{

//Do something if needed

return the\_Desired\_Value;

}

}

A property is a member that represents data.

A property appears very much like a field. However, a property is a function member, and it does not necessarily allocate memory of data.

A property is a name set of two methods called accessors: set and get.

Method: the function inside the class

Constructor: a special method that is used to initialize the state of a class instance. It is executed whenever a new instance of a class is created. It does not have any return type. \*\*\*\* The name of the constructor is the same as that of the class.

Access modifier: if not specified, all are private automatically. Make public if needed.

OOP concepts:

Encapsulation: information hiding, making them private. Changes made would not influence other codes outside of class.

Inheritance: Types can be derived from other types. That is, a type can be a sub type. So, we can reuse the components by deriving needed parts from previous work.

Polymorphism (many forms/shapes): It is the ability of an object to take on many forms.

Subroutine: consists of the instructions for performing some tasks, grouped together as a unit and given a name.

Method overloading:

Method overloading allows us to create multiple implementations of a method with the same name. Compilers choose which one according to the parameters.

String: non-simple predefined type

Strings are immutable and cannot be modified. Changing the string would result in a new string object.

Stringbuilder: Mutable alternative

Graphical user interface, text

Description automatically generated

Lists:

A screenshot of a computer

Description automatically generated

Week 3:

Static checking: check before compiling.

Dynamic checking: find bugs in executing

No checking: check yourself.

Week 4:

Specification: written to tell what the program does

Precondition: indicated by the keyword requires

Postconditions: indicated by the keyword effects

C# allows references to objects and arrays to take “null”, which means that the reference does not point to any object.

Operation on them would be through NullReference Exception. It is not the same as empty string or arrays. Null values are implicitly disallowed as parameters and return values.

Unit test: involves an Arrange/Act/Assertion pattern

Arrange: setup part in which we initialize and select the test case.

Act: call the method to be tested

Assertion: compare the expected to the results

Week 5:

Documentation comments: /// only used to describe the type and type member… (machine readable and human readable)

XML (extensible markup language) Tags start with < and end with>

Tag: summary, param, returns, exception…

Text, letter

Description automatically generated

Intro to exception:

Try {}

Catch (some parameter) {}

Finally {} //sth to do if not caught by anyone, have to be the last one

Exception is a run time error.

We can throw an exception statement when needed

Week 6:

Static: methods with static can be called directly. Does not have to be {type.method().}

Comparing statement:

More specific statements are stronger:

Specification A is stronger than or equal to specification B if

A’s precondition is weaker than or equal to B’s

A’s postcondition is stronger than or equal to B’s, for the states that satisfy B’s precondition.

Deterministic: the output is determined when presented satisfying precondition

Under-deterministic: multiple output would be true for the presented satisfying precondition

Declarative spec: specification do not give details of intermediate steps. Instead, they just give properties of the final outcome, and how it is related to the initial state.

Operational spec: gives a series of steps that the method performs.

HashSet<T>: unordered, no duplicates

Week 7:

Testing:

Software validation:

1. Testing: run the program
2. Code review: Someone else go through it
3. Formal reasoning (verification): beyond the scope

Exhaustive testing is infeasible: Can not do all the cases

Haphazard testing: random testing, not good

Statistical testing: test for trend, not good for software. All behaviours are discrete

Blackbox testing: only know input and output

Whitebox testing: know the steps in the box

Coverage:

Statement coverage

Branch coverage

Path coverage

A good test suite covers different aspects of program behaviour. Small enough to run but big enough to sufficiently validate the program

Partitioning: divide the input space into subdomains

Subdomains should completely cover the input space

Each subdomain covers sets of similar inputs on which the program has similar behaviour. We also consider special or corner cases

Choose one good test case from each subdomain

AVOID: magic numbers

Tester’s aim is to fail the program

Version control: keep the previous codes for later use

Local: keep on your own computer

Centralized: central server

Distributed: each user keeps a full copy

Terminology:

Repository: a storage place for the project files (and the history)

Head: the current version

Master: the default branch name

Week 8:

Delegate allow u to pass a function/method as a variable. All functions with the same parameter and return type can be delegated the same way.

Alias: multiple reference to the same mutable object

Passing by parameters:

Pass value-type by value

Pass value-type by reference

Pass reference-type by value

Pass reference type by reference

Pass by reference = ref/out

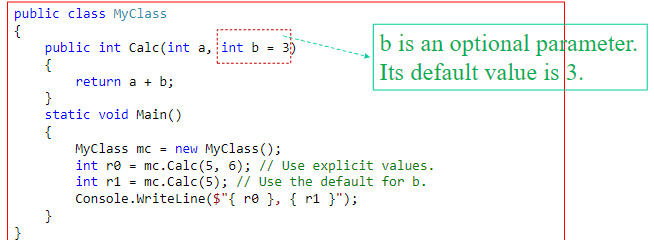
Diagram

Description automatically generated

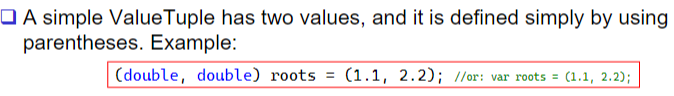
W9:

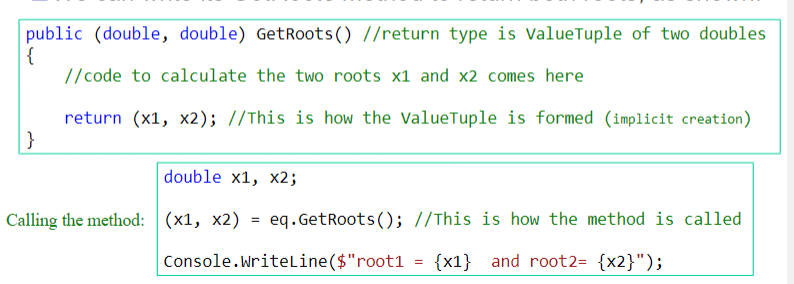
Local method: Scope of the method would be inside the method. Although it is ok to put it at the bottom, but it is a good idea to put it at the top.

Optional parameter:

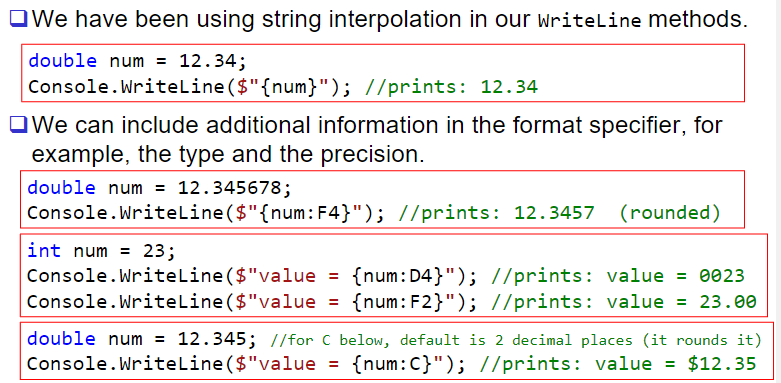


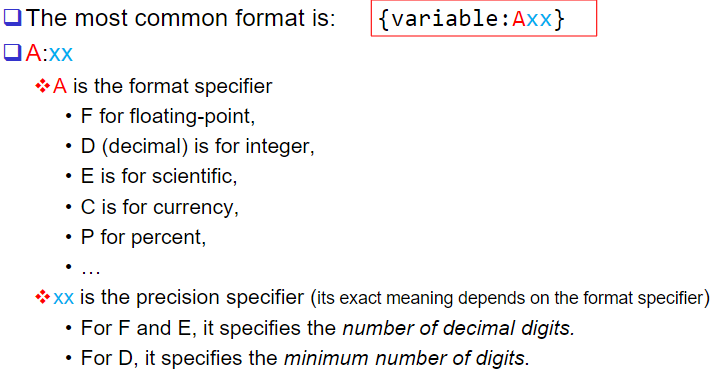
Value tuples: a form of data structure that holds an ordered list of up to seven elements, where the elements can be of different data types. It immutable





Formatting numeric strings:





Week 10:

Mutability:

Try not to modify the original input if not asked to

Alias:

Two different references pointing to the same thing. Changes on either one would change the value. So, for mutable references, make sure what you are changing is not an alias of sth else.

Recursion:

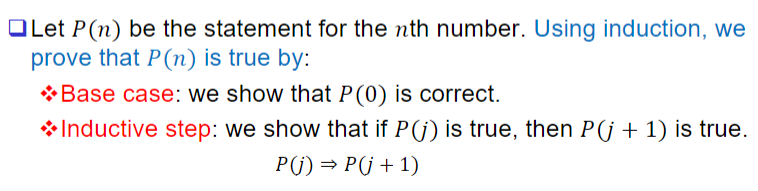
Method that calls itself consistently until terminating point. Decompose the problem until sth that we know.

Default stack overflow is 19274.

Problem of recursion is that it uses too much of memory. It needs to store all the data in each recursion step.

So, we only use recursion in naturally recursive tasks.

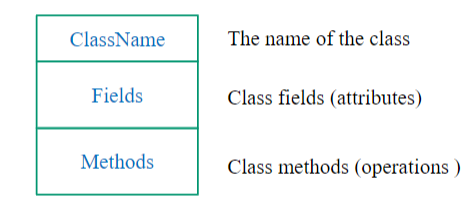
Induction:



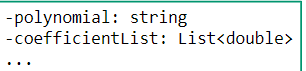
Week 12:

UML (unified modeling language):

Used when we only want to identify the class in UML, but we would like to hide/abstract its members.

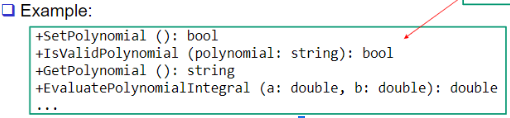


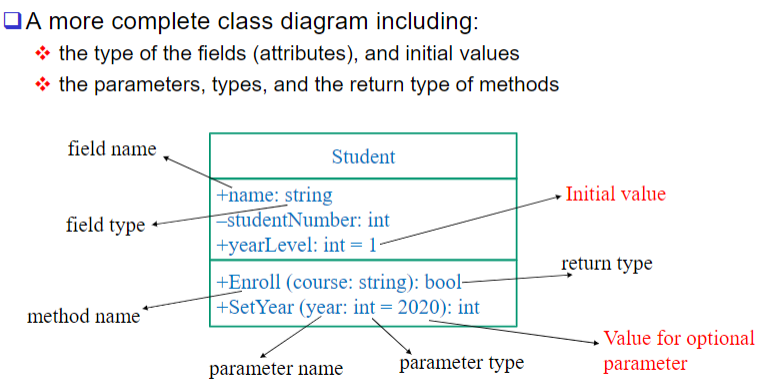
Each field appears on one line. [+/-fieldName: fieldType]

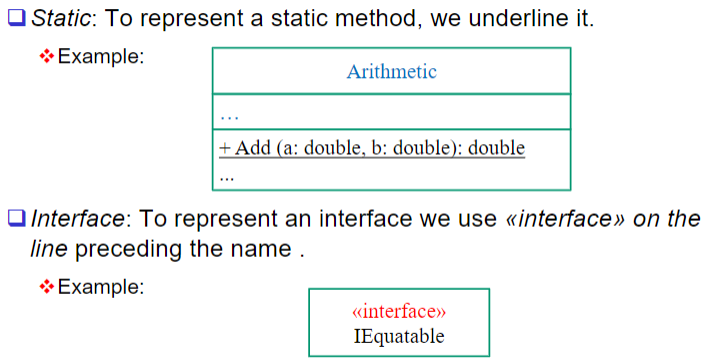
E.g., 

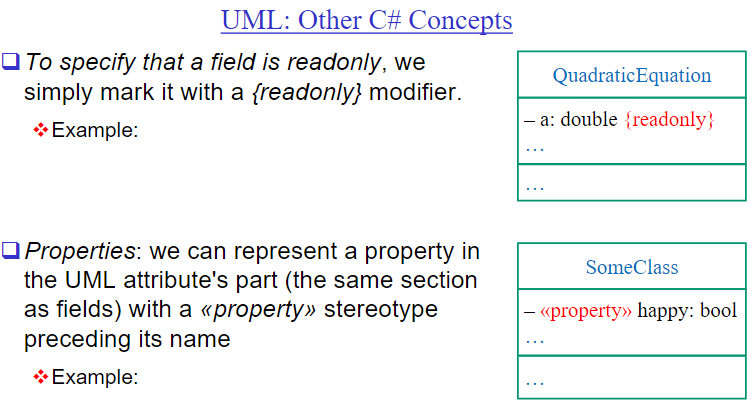
Each method appears on one line.

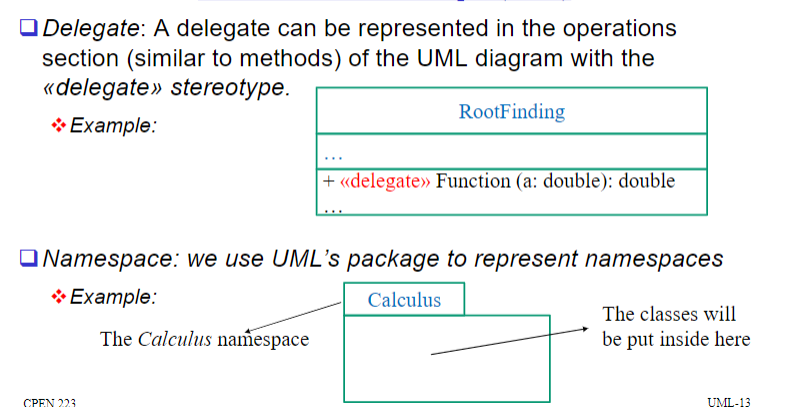
[+/-methodName (parameter: type /\*separated by comma\*/]): return type]

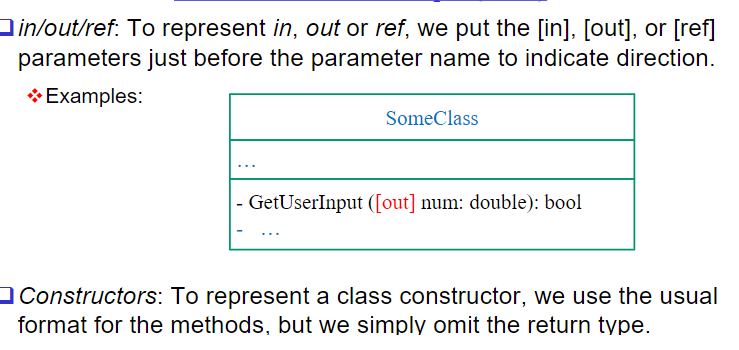


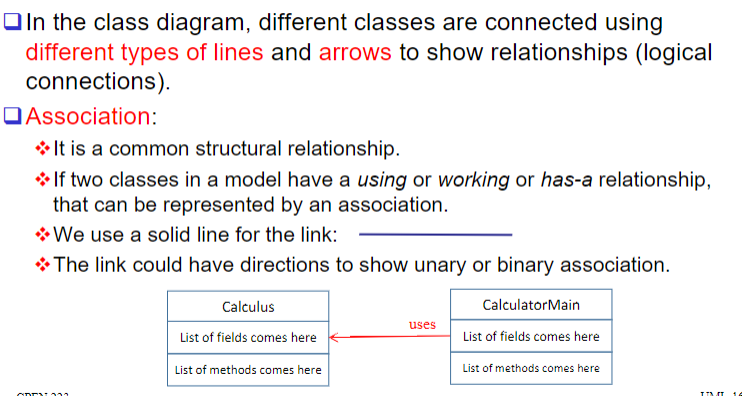


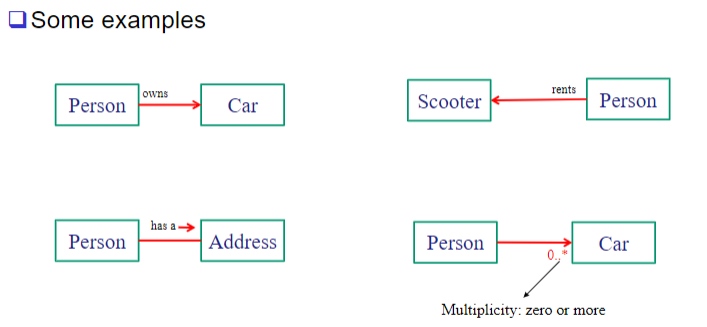


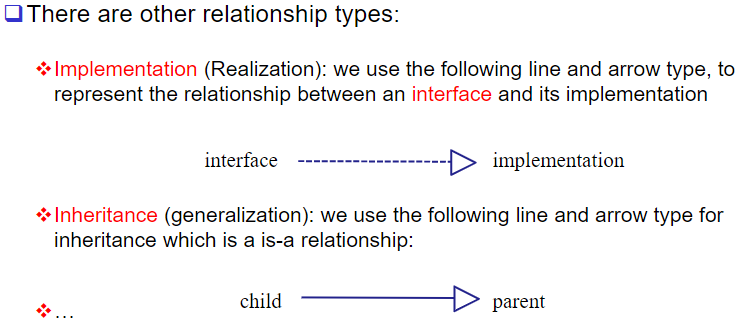












Interface:

Interface is a reference type provided in C#. It defines a contract.

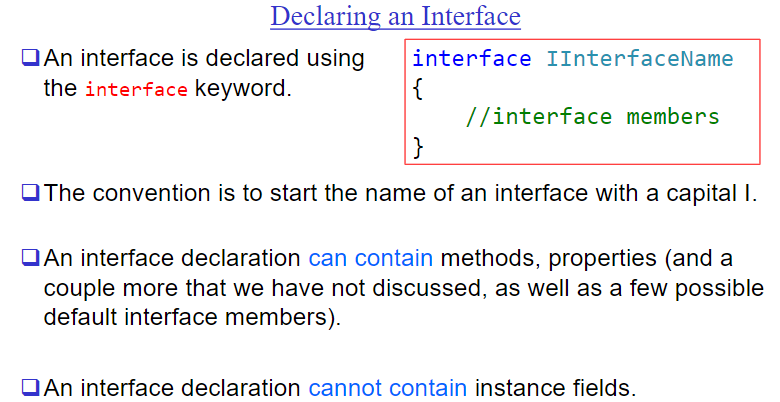
An interface declaration may look like a class’s but

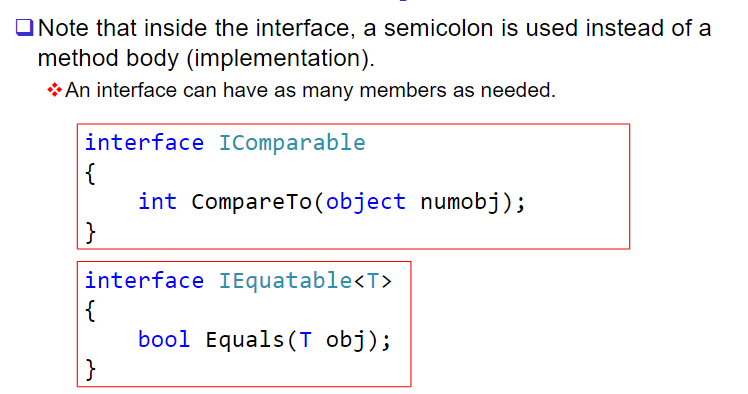
It can only define function members and not fields

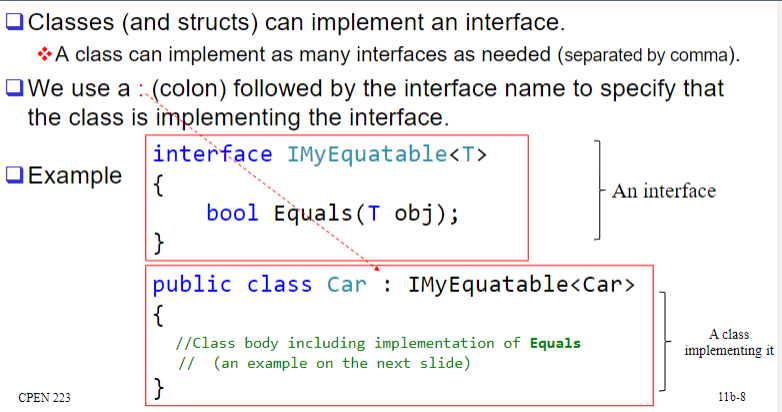
The function members do not (normally) have an implementation (in another word, they are implicitly abstract). The implementation is postponed to later, for example, to a class that implements that interface.

Interface members are always implicitly public.

A class can implement multiple interfaces







Different classes can implement the same interface and the interface can be used interchangeably.

Why interface:

One of the useful mechanisms for expressing abstract data type.

Its purpose is to specify the contract for the client

We can also easily have multiple different representations of the abstract data type coexist in the same program.

Allow performance trade-off

Flexibility in providing invariants

Documentation for both the compiler and human

Methods with intentionally underdetermined specifications

…

Week 13:

Abstract Data Type (ADT): enable us to separate how we use a data structure in a program from the particular form of the data structure itself.

Abstraction: Omitting or hiding low-level details with a simpler, higher-level idea.

Modularity: Dividing a system into components or modules, each of which can be designed, implemented, tested, reasoned about, and reused separately from the rest of the system.

Separation of concerns: Making a feature (or “concern”) the responsibility of a single module, rather than spreading it across multiple modules.

What made abstract types new and different was the focus on operations: the user of the type would not need to worry about how its values were actually stored, in the same way that a programmer can ignore how the compiler actually stores integers. All that matters is the operations.

Classifying operations:

1. Creators:

A creator may take any object as an argument, but not an object of the type being constructed. E.g., constructors: 无中生有

1. Producer:

Create new objects from old objects of the same type

1. Mutator:

Change object

1. Observer:

Take objects of the abstract type and return objects of a different type

Representation: the actual data structures that are used internally to support operations on an abstract data type.

Representation independence: the use of an abstract type is independent of its representation. E.g., List behaves the same no matter what the type is in there.

If working with a variety of the same reference data, one can use multiple references to the same data with different values of field to distinguish the different expectations of the same data.

Invariant: a characteristic of a program/object that is always true.

Make the invariant true in the initial state of the object (creators and producers).

Ensure that all changes to the object keep the invariant true (Mutators and observers).

Abstraction function: captures the designer’s intent in choosing particular representation.

文本

描述已自动生成

Representation Invariant:

RI is a statement of a characteristic/property of the rep that all legitimate objects/instances of a type satisfy.

图形用户界面, 文本, 应用程序

描述已自动生成

