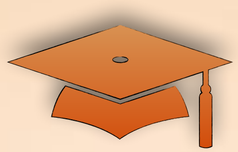
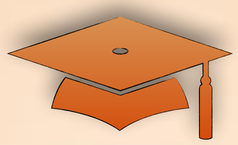
The Complete C# Developer Course



Ahmad Mohey | Full Stack Developer

E-mail : ahmadmohey@gmail.com
Twitter : ahmadmohey85

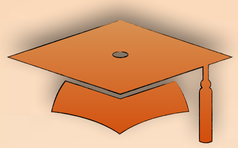
Object-oriented Programming Part 1



Ahmad Mohey | Full Stack Developer

E-mail : ahmadmohey@gmail.com
Twitter : ahmadmohey85

Object-oriented Programming Part 2

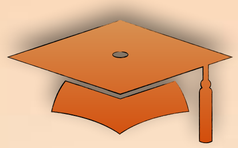


Sealed classes

Ahmad Mohey | Full Stack Developer

E-mail : ahmadmohey@gmail.com
Twitter : ahmadmohey85

Object-oriented Programming Part 2



Static classes

Ahmad Mohey | Full Stack Developer

E-mail : ahmadmohey@gmail.com
Twitter : ahmadmohey85

Static classes

- You can only have static members.
- You cannot create an instance of it.
- They are sealed

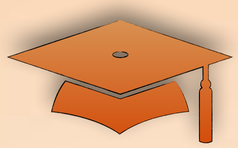
Object-oriented Programming Part 2

Nested classes

Ahmad Mohey | Full Stack Developer

E-mail : ahmadmohey@gmail.com
Twitter : ahmadmohey85

Object-oriented Programming Part 2

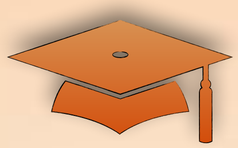


Partial classes

Ahmad Mohey | Full Stack Developer

E-mail : ahmadmohey@gmail.com
Twitter : ahmadmohey85

Object-oriented Programming Part 2

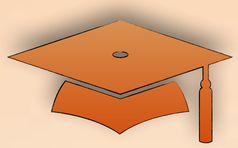


Namespaces

Ahmad Mohey | Full Stack Developer

E-mail : ahmadmohey@gmail.com
Twitter : ahmadmohey85

Object-oriented Programming Part 2

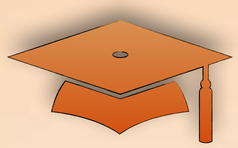


Struct

Ahmad Mohey | Full Stack Developer

E-mail : ahmadmohey@gmail.com
Twitter : ahmadmohey85

Object-oriented Programming Part 2



Classes VS Structs

Ahmad Mohey | Full Stack Developer

E-mail : ahmadmohey@gmail.com
Twitter : ahmadmohey85

Classes VS Structs

Class

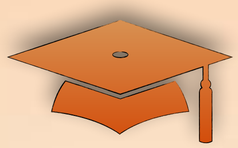
- ✓ Declared with class keyword
- ✓ Supports inheritance
- ✓ User-defined constructors can be implemented
- ✓ Data members can be initialized in the class definition
- ✓ Reference type (Heap)

Struct

- ✓ Declared with struct keyword
- ✓ Doesn't Support inheritance
- ✓ User-defined constructors can't be implemented
- ✓ Data members can't be initialized in the struct definition
- ✓ Value type (Stack)

The majority of types in a framework should be classes, but if instances of the type are small and commonly short-lived or are commonly embedded in other objects define a struct.

Object-oriented Programming Part 2



Enumerations

Ahmad Mohey | Full Stack Developer

E-mail : ahmadmohey@gmail.com
Twitter : ahmadmohey85

Object-oriented Programming Part 2

Country exercise

Ahmad Mohey | Full Stack Developer

E-mail : ahmadmohey@gmail.com
Twitter : ahmadmohey85

Country Exercise

Country class

Child of a base class called world, and world class itself has couple of variables called planet name and continent and properties to access these variables

Variables : country name, capital, languages , currency
Properties : Country name, Capital, Languages and Currency
Constructor : initialize country name and capital
Methods : two overloaded methods to display welcome message

Struct: To define the first and second language for each country
Enums : 3 different enums for the continents, currencies and languages

Country Exercise

Here is what is required (1 of 2):

1. Define namespace called ***WorldNamespace***
2. Create a base class called ***World***
3. Create non-accessible (outside class) variable called ***planetName***
4. Define enumerations called ***Continents*** (with the continents names)
5. Create non-accessible (outside class) instance of enum Continents called ***Continents***
6. Create a property to allow access to ***planetName*** and ***Continents***
7. Define two more enums called ***Currencies*** and ***Languages***
8. Define a child class called ***Country*** derived from ***World***
9. Define non-accessible (outside class) four variables in class ***Country*** which are ***countryName, capital, countryLanguages and currency***
10. Define a struct called ***CountryLanguages*** which has two non-accessible (outside class) variables called ***firstLanguage*** and ***secondLanguage*** and two properties called ***FirstLanguage*** and ***SecondLanguage*** to allow access to ***firstLanguage*** and ***secondLanguage***

Country Exercise

Here is what is required (2 of 2):

11. Create four properties in class ***Country*** named ***CountryName, Capital, CountryLanguages and Currencies*** to give access to four variables ***countryName, capital, countryLanguages and currency***
12. Create a constructor to initialize properties ***CountryName and Capital***
13. Define two overloaded methods called ***SayHi*** (which says Hi) one with no arguments and the other with one string arguments to mention the country
14. In the main method create two instances of ***Country*** class name them ***countryOneInstance*** and ***countryTwoInstance***
15. Create instance of the struct ***CountryLanguages*** called ***countryLanguages***
16. Assign two different languages using ***Languages*** enum
17. Assign the instance ***countryLanguages*** to ***countryOneInstance.CountryLanguages***
18. Set ***planetName*** to “earth” and ***countryName*** to “any country”
19. Set ***Currency*** and ***Continents*** to corresponding enums values
20. Call SayHi methods and display values of ***countryOneInstance***

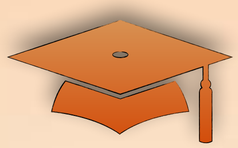
Object-oriented Programming Part 2

Country exercise solution

Ahmad Mohey | Full Stack Developer

E-mail : ahmadmohey@gmail.com
Twitter : ahmadmohey85

Object-oriented Programming Part 2

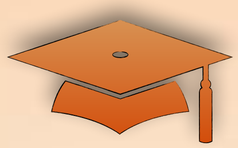


this keyword

Ahmad Mohey | Full Stack Developer

E-mail : ahmadmohey@gmail.com
Twitter : ahmadmohey85

Object-oriented Programming Part 2



Interfaces

Ahmad Mohey | Full Stack Developer

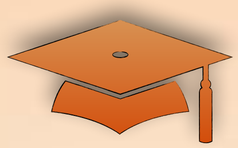
E-mail : ahmadmohey@gmail.com
Twitter : ahmadmohey85

Interfaces

An interface contains definitions for a group of related functionalities that a class or a struct can implement.

Think of it as contract that all the classes inheriting the interface should follow. The interface defines the '*what*' part of the contract and the deriving classes define the '*how*' part of the contract.

Object-oriented Programming Part 2



Interfaces VS Abstract Classes

Ahmad Mohey | Full Stack Developer

E-mail : ahmadmohey@gmail.com
Twitter : ahmadmohey85

Interfaces VS Abstract Classes

Interface

Abstract Class

Similarities

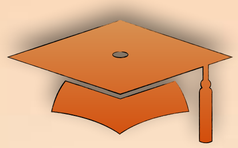
- Can't be instantiated directly
- Must implement all its members
- Can contain events, methods, and properties.

Differences

- Can’t have method implementations
- Allow multiple inheritance
- Can’t have access modifiers, everything is public
- Can’t contain variables

- Can have method implementations
- Doesn’t allow multiple inheritance
- Can contain access modifiers
- Can contain variables

Object-oriented Programming Part 2



Exception Handling

Ahmad Mohey | Full Stack Developer

E-mail : ahmadmohey@gmail.com
Twitter : ahmadmohey85

Exception Handling

An exception is a runtime error that happens during the execution of a program.

Exceptions provide a way to transfer control from one part of a program to another.

Exception Handling

Exception handling is built upon these keywords: *try*, *catch* and *finally*

***Try*: try block contains a block of code, exceptions is expected to happen if we run it.**

***Catch*: program catches an exception with an exception handler at the place in a program where you want to handle the problem.**

***Finally*: finally block is used to execute a block of code, whether an exception is thrown or not thrown.**

Exception Handling

```
try
{

}
catch (SomeSpecificException ex)
{

}
```

Exception Handling

```
try
{

}
finally
{

}
```

Exception Handling

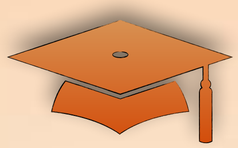
```
try
{

}
catch (SomeSpecificException ex)
{

}
finally
{

}
```

Object-oriented Programming Part 2

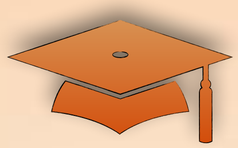


Exception exercise

Ahmad Mohey | Full Stack Developer

E-mail : ahmadmohey@gmail.com
Twitter : ahmadmohey85

Object-oriented Programming Part 2



Composition

Ahmad Mohey | Full Stack Developer

E-mail : ahmadmohey@gmail.com
Twitter : ahmadmohey85

Composition

Inheritance

Can be defined as "is-a" relationship between objects
Car is a vehicle

```
class Vehicle
{

}
class Car : Vehicle
{

}
```

Composition

Composition

Can be defined as "part-of" relationship between objects
Engine is a part of Car

```
class Engine
{

}
class Car
{
    Engine engine = new Engine();
}
```

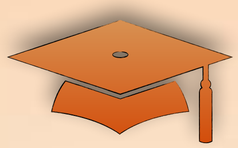
Composition

- **The lifetime of the child instance is dependent on the owner or the parent class instance. Child objects does not have their lifecycle without parent object.**

- **If a parent object is deleted, all its child objects will also be deleted.**

The branches do not have their independent life cycle, it depends on the bank object's life (parent object).

Object-oriented Programming Part 2



Aggregation

Ahmad Mohey | Full Stack Developer

E-mail : ahmadmohey@gmail.com
Twitter : ahmadmohey85

Aggregation

Aggregation

Can be defined as "has-a" relationship between objects

Teachers and addresses, a teacher must have an address associated with him/her
But if you consider its vice versa then it would not make any sense

Aggregation

Teacher ***Has-An*** Address (Has-a relationship between teacher and address)
School ***Has-An*** Address (Has-a relationship between school and address)
Student ***Has-An*** Address (Has-a relationship between student and address)

```
class Address
{
}
```

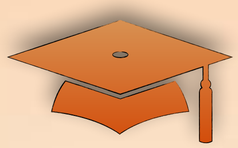
```
class TeacherClass
{
   Address address = new Address();
}
```

```
class SchoolClass
{
   Address address = new Address();
}
```

```
class StudentClass
{
   Address address = new Address();
}
```

- There is an ownership of the instances
- No lifetime dependency on parent class

Object-oriented Programming Part 2



Association

Ahmad Mohey | Full Stack Developer

E-mail : ahmadmohey@gmail.com
Twitter : ahmadmohey85

Association

Association

Can be defined as “using" relationship between objects

Teacher and student, a teacher can be teaching many students and student maybe learning from many teachers

- **There is no ownership**
- **No lifetime dependency**

Association, Aggregation and Composition

Association

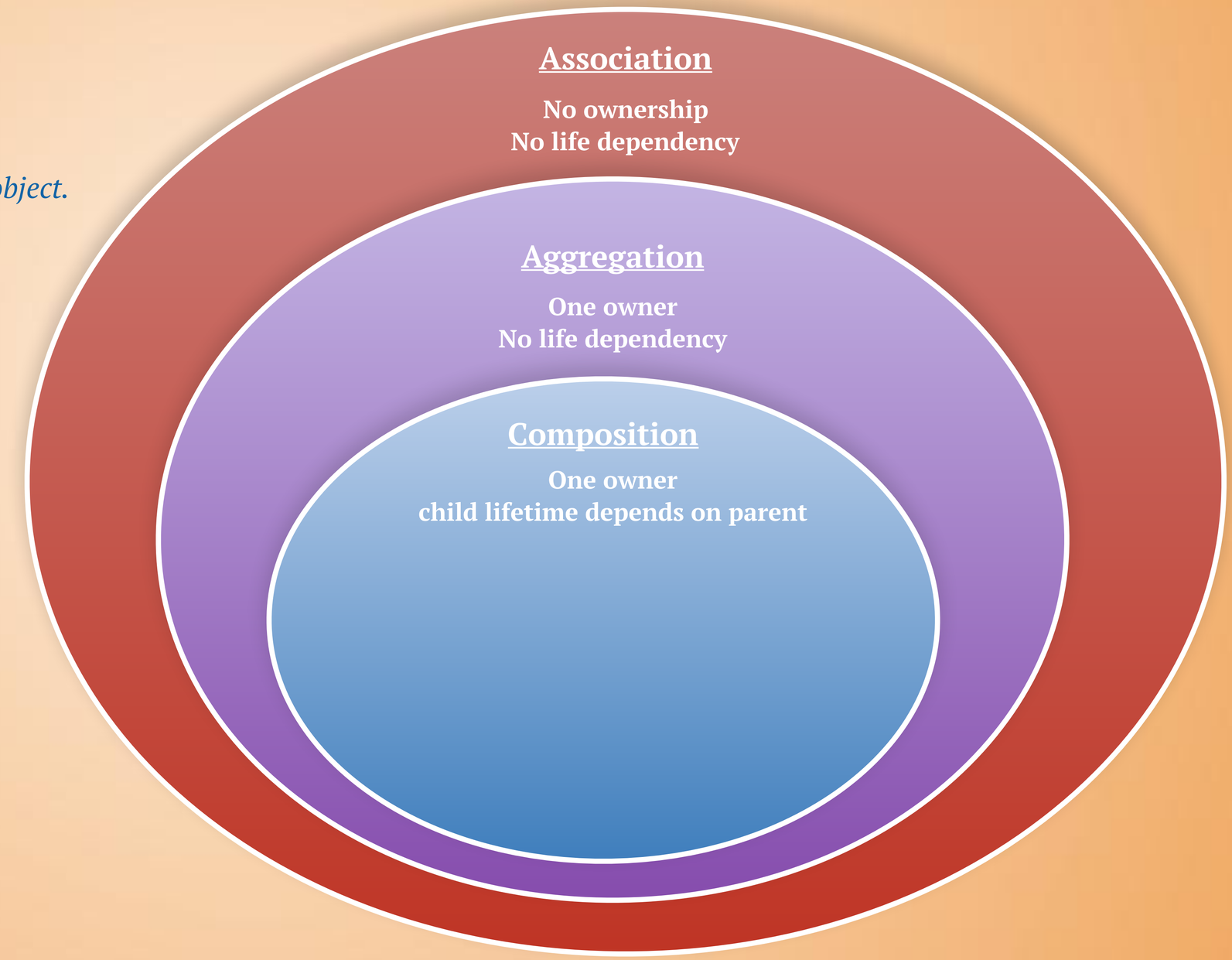
- *It is a relationship between objects.*
- *Usually defined as “is-a” or “using” relationship.*
- *Both objects have independent life-cycle.*
- *Each object owns their actions and will not affect other object.*

Aggregation

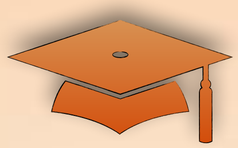
- *Specialized form of Association.*
- *Usually defined as “has-a” relationship.*
- *Each object has an independent life-cycle.*
- *Ownership between objects.*

Composition

- *Specialized form of Aggregation.*
- *Usually defined as “part-of” relationship.*
- *Child Object has dependent life-cycle.*
- *Ownership between objects.*



Object-oriented Programming Part 2



Assignments(7, 8)

Ahmad Mohey | Full Stack Developer

E-mail : ahmadmohey@gmail.com
Twitter : ahmadmohey85

Assignments

Assignment No.7: (Movie struct)

- Movie name, rate, releaseDate, genres, languages, countries
- Genres, languages and countries are enum and instances is allowed to have multiple values
- Create a method in the struct to display the following output

Output

Movies : Wild Tales
Rate : 8.1 out of 10
Released on: 8/21/2014
Languages: Spanish
Countries : Argentina, Spain
Genres : Comedy, Drama, Thriller

http://www.imdb.com/

Assignments

Assignment No.8: (Dog interface)

- **Base class called Animal (name, birth date and origin)**
- **Interface called Dog contains methods like eat(), Drink(), Sit(), SayHi()..etc**
- **Class GermanShepard derived from Animal and implements Dog interface**
- **GermanShepard class has the following (gender, weight, security guard, training, size)**
- **Create overloaded methods one with no parameters and one with dog name parameter**

Output

The dog name is Max from Germany, was born on 10/10/2017
Max is a Male and weights 32.54 kilos and its size is Large

Max is sitting
Dog is eating
Hiii "in dogs language :D"

Object-oriented Programming Part 2

- ✓ Sealed classes
- ✓ Static classes
- ✓ Nested classes
- ✓ Partial classes
- ✓ Namespaces
- ✓ Structs
- ✓ Structs VS Classes
- ✓ This keyword
- ✓ Interfaces
- ✓ Interfaces VS Abstract classes
- ✓ Exception handling
- ✓ Composition, Aggregation and Association

Ahmad Mohey | Full Stack Developer

E-mail : ahmadmohey@gmail.com
Twitter : ahmadmohey85