C# Advanced – Class 2

Material recap: Classes, Inheritance

New material: Abstract classes, Polymorphism, Interfaces

Trainer: Miodrag Cekikj – cekicmiodrag@gmail.com

Assistant: Andrej Chichakovski – andrejchichak@gmail.com

SEAVUS EDUCATION *and* DEVELOPMENT CENTER

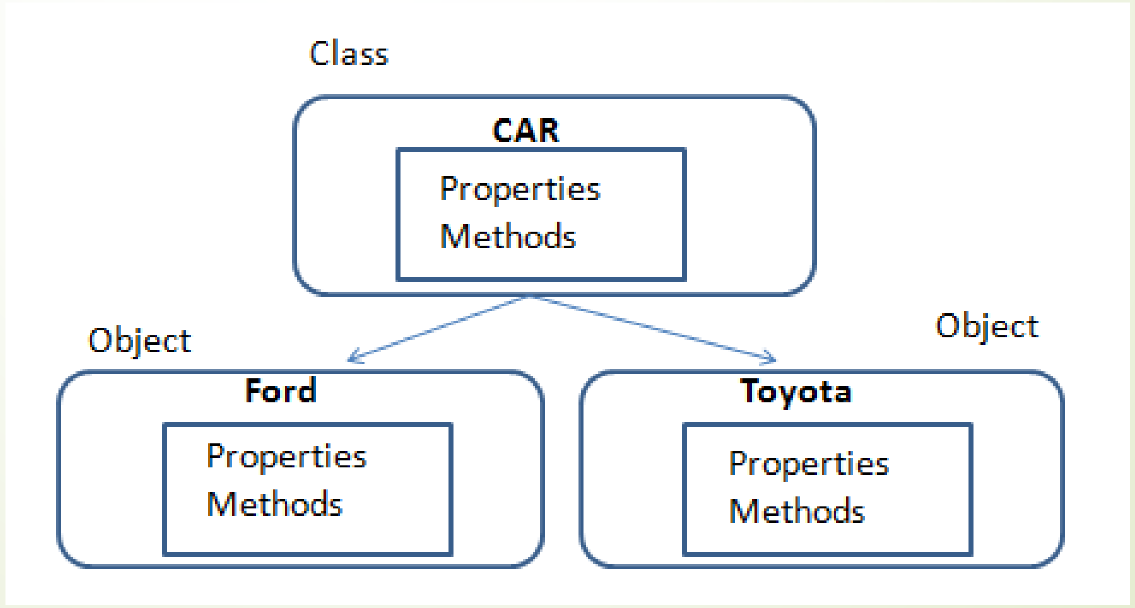
Code academy @ Skopje, 2019

Agenda

- Classes (recap)
- Inheritance
- Abstract classes
- Polymorphism
- Interfaces

Classes (recap)

- The class is the **fundamental building block of code** in the object-oriented programing. When an object is ***instantiated***, it has all of the methods, properties and other behavior defined within the class.
- In the real world the classes describe the objects, and the objects are the instances of them with the **properties** and **methods** defined within the classes.



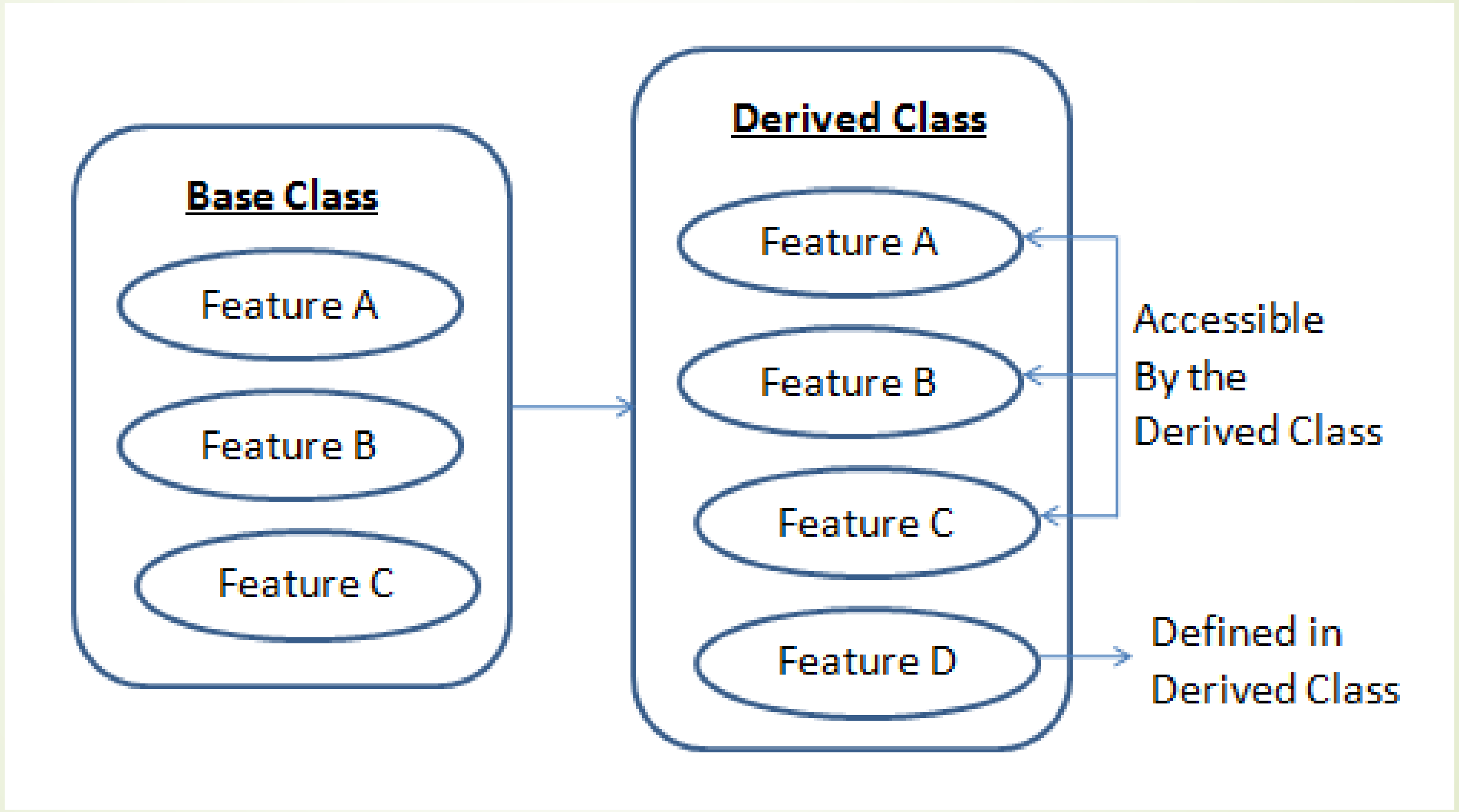
Inheritance

- **Inheritance** is the ability to **create a class from another class**, the **"parent" class**, **extending the functionality** and **state** of the parent in the derived, or **"child" class**. It allows derived classes to override methods from their parent class.
- **Inheritance is one of the pillars of object-orientation**. It is the mechanism of designing one class from another and is one of the ideas for <u>**code reusability**</u>, **supporting the concept of hierarchical classification**. <u>C# programs consist of classes, where new classes can either be created from scratch or by using some or all properties of an existing class</u>.
- **Another feature related to inheritance and reusability of code is polymorphism**, which permits the same method name to be used for different operations on different data types. Thus, C# supports code reusability by both features.
- Important characteristics of inheritance include:
  - A derived class extends its base class. That is, it contains the methods and data of its parent class, and it can also contain its own data members and methods.
  - The derived class cannot change the definition of an inherited member.
  - All other members of the base class are inherited.
  - The accessibility of a member in the derived class depends upon its declared accessibility in the base class.
  - A derived class can override an inherited member.

Accessors visibility matrix

visibility / keyword	Containing Classes	Derived Classes	Containing Assembly	Anywhere outside the containing assembly
public	yes	yes	yes	yes
protected internal	yes	yes	yes	no
protected	yes	yes	no	no
private	yes	no	no	no
internal	yes	no	yes	no

Inheritance



Abstract classes

- An ***abstract class*** provides all of the characteristics of a concrete class except that it <u>**does not permit objects of the type to be created**</u>.
- An abstract class simply **defines members** (properties and methods) that are used for inheritance, defining the functionality of its child classes.
- These members may themselves be **abstract**, effectively declaring a placeholder that must be implemented by subclasses. Members may also be **concrete**, including real functionality, and may be marked as *virtual* to support polymorphism via method overriding.
- PROVIDE **common behavior** with forcing to **follow already implemented design** (ex. Custom frameworks, Power Point application, etc.)
- FileStream in Object Browser (Visual Studio)

Abstract classes – GOLD Rules

- When a **class member is declared as abstract**, **that class needs to be declared as abstract as well**. That means that class is not complete.
- In **derived classes**, we **need to override the abstract members in the base class**.
- In a derived class, we need to **override all abstract members of the base class**, otherwise that derived class is going to be abstract too.
- **Abstract classes cannot be instantiated**.

Polymorphism

- **Polymorphism** is often referred to **as the third pillar of object-oriented programming**, after encapsulation and inheritance (by Microsoft Doc).
- **Polymorphism** ⇔ “Many-shaped”

- **Poly** ⇔ “Many”
- **Morph** ⇔ “Form”
- **Polymorphism** ⇔ “Many Forms”
- Ex: **Draw** many different **Shapes**

Interfaces

- An **interface** is a code structure that is **similar to an abstract class** that has **no concrete members (properties and methods)**. An interface can contain public members such as properties and methods but these members **must have no functionality**.
- The interfaces define items that **must be made concrete** within all classes that ***implement*** the interface. **This means that an interface can be used to define what a class must do, but not how it will achieve it**.
- *The "I" prefix for an interface name is a recognized* ***naming convention***. *E.g* IVehicle, IAnimal, IPerson, etc.
- **Language construct** similar to class, but <u>fundamentally different</u>..?

Polymorphism types (DEMOs)

- **Polymorphism means that a operation can also be applied to values of some other types** (more definitions for polymorphism).

There are multiple types of Polymorphism:

- **Static (or ad-hoc) polymorphism:**

  Demo: Polymorphism-Static

- **Dynamic (Subtyping) polymorphism :**

  Demo: Polymorphism-Dynamic

- **Polymorphism with abstract classes**

  Demo: Polymorphism-Abstract

- **Polymorphism with virtual methods**

  Demo: Polymorphism-Virtual