Exercise: Encapsulation and Polymorphism

This document defines an in-class exercise from the "OOP" Course @ Software University.

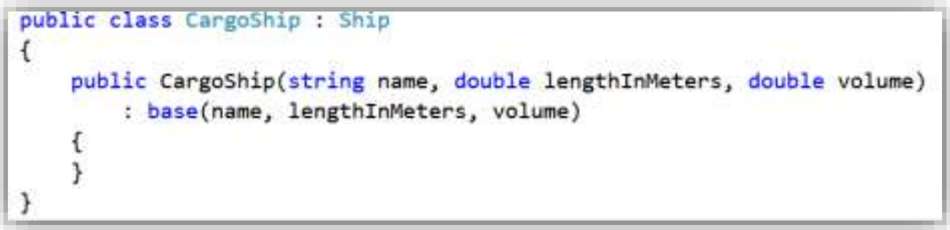
Problem 2. Using Polymorphism

Abstraction gives us the ability to define abstract actions. With polymorphism, we can specify how each of these actions is implemented in specific classes. Then, we can perform these actions and the appropriate overridden method will be called.

For this problem, you'll be using the **Polymorphism.sln** solution. There are several classes in the project, one engine performing some interactions, and ships. Your task is to reduce code duplication and use overridden methods in order to achieve better abstraction.

Step 1. Put All Common Features in a Parent Class

All of the ships have some **common** fields and properties (name, length, volume). They can all be put in the Ship class and reused by child classes. Example:



```
public class CargoShip : Ship
{
    public CargoShip(string name, double lengthInMeters, double volume)
        : base(name, lengthInMeters, volume)
    {
    }
}
```

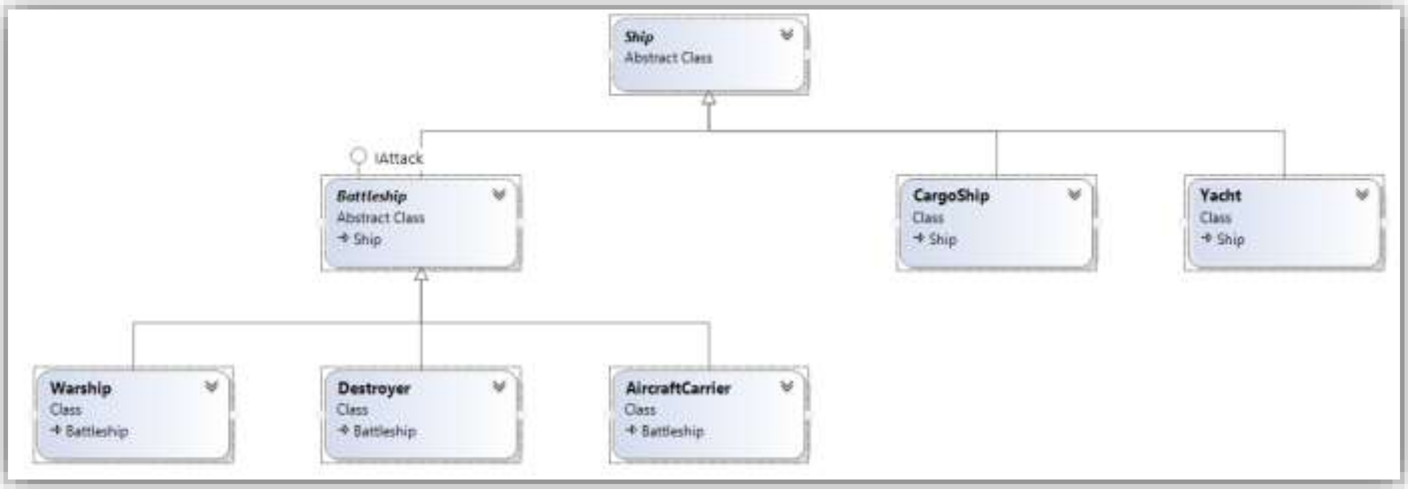
Step 2. Add Interface and Abstract Class for Battleships

Some of the ships are special, they can **attack** other ships. These are battleships. An action like this calls for an **interface IAttack** defining the method **Attack**. Leave the method **void** for now; it should accept one parameter – a target of type **Ship**.

Add also a new **abstract class Battleship** which inherits Ship and implements IAttack.

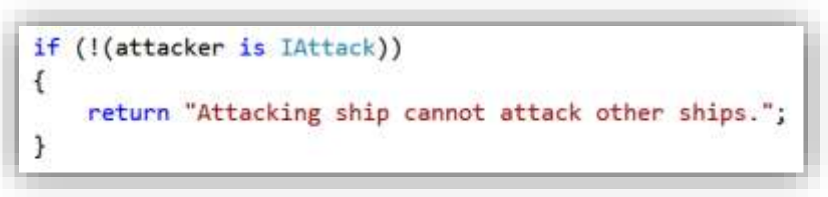
Step 3. Complete the Inheritance Hierarchy

All ships that are able to attack should inherit the new abstract class Battleship. The **IsBattleship** property in the Ship class will be redundant when you're done, remove it.



Step 4. Study the Engine

To simplify your task without breaking the game logic, you'll need to get familiar with the Engine. It's pretty simple – it has a list of ships and performs several attacks by picking random ships from the list, one attacker and one defender. It performs several checks – non-battleships cannot be attackers, so this returns an error message; destroyed ships cannot attack and cannot be attacked. Now that the IsBattleship property is gone, you can check if a ship can attack by checking the type with the "**is**" operator, e.g. "if (attacker is IAttack)" or "if (attacker is Battleship)". Example:



```
if (!(attacker is IAttack))
{
    return "Attacking ship cannot attack other ships.";
}
```

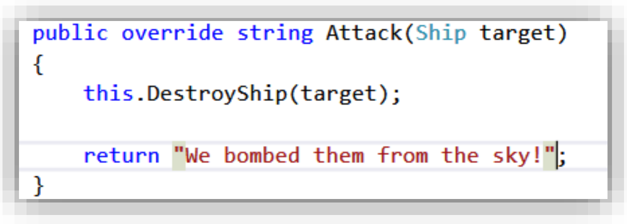
If the attacker is a battleship and both ships are intact, the attack goes normally.

What is an attack? It **destroys** the target ship and **returns a message** as a result depending on the attacking ship. This means the **Attack method should return a string**. Modify the IAttack interface to reflect the change.

Step 5. Implement and Override the Attack Method

Since an attack always destroys the target ship, this means you can implement a **DestroyTarget** method in the **Battleship** class and reuse it in child classes. What should be the access modifier of this method? What type of arguments should it accept?

In each child of Battleship, within the Attack method, the DestroyTarget method should be called and a string should be returned. Example (the AircraftCarrier class):



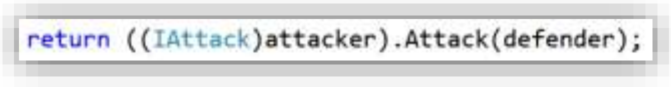
```
public override string Attack(Ship target)
{
    this.DestroyShip(target);

    return "We bombed them from the sky!";
}
```

This will allow us to call the Attack method of the attacker directly and store the result in a variable which the engine will print. Currently, the engine performs explicit checks to see what the attacker's type is and decides what message to return, which is hard to maintain (what happens if we add more ships?).

Step 6. Modify the Engine

Now that the Attack method is overridden, you can call it in the engine. Because you're working with objects of type Ship, you'll first need to cast the attacker to IAttack or Battleship in order to call the Attack method (Ship does not contain a definition of the Attack method).

Then, all you need to do is return the result of the Attack method from the SimulateAttack method (replace the switch-case with the following line of code:



```
return ((IAttack)attacker).Attack(defender);
```