# Lab: Object Communication and Events

This document defines the exercises for [C# OOP Advanced" course @ Software University](https://softuni.bg/courses/csharp-oop-advanced-high-quality-code).

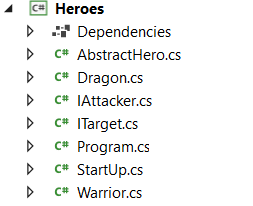
You can check your solutions here: [Judge](https://judge.softuni.bg/Contests/982/Object-Communication-and-Events-Lab) .

# Part I: Chain of Responsibility, Command Design Pattern

## Resources

**NOTE**: You need a public StartUp class with the namespace **Heroes**.

You are given a file with some classes. Place them in a new project and get familiar with them.



## Logger - Chain of Responsibility

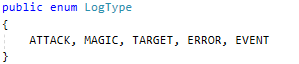
Create a **Chain of Responsibility** Logger and provide:

* enum LogType
  + values - ATTACK, MAGIC, TARGET, ERROR, EVENT
* interface IHandler
  + void Handle(LogType, String)
  + void SetSuccessor(Handler)
* Concrete loggers that log messages to console:
  + CombatLogger
  + EventLogger

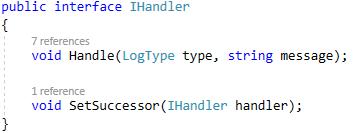
Log messages in format (**"TYPE: message"**)

### Solution

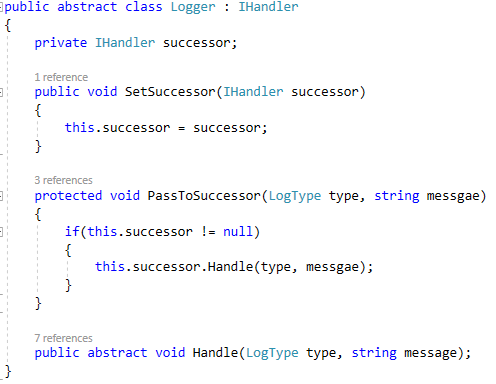
Create enum LogType



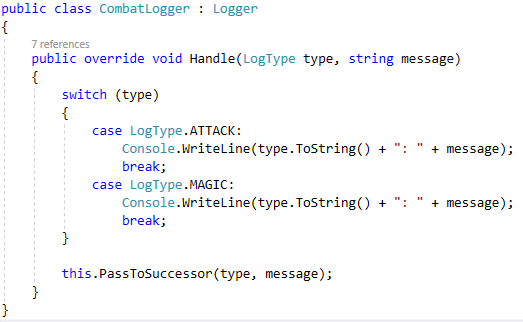
Create IHandler interface



You can create an abstract logger, so you can abstract some of the functionalities



Create a concrete logger that extends Logger



Test the logger through you client.(In the Main method)

Don't forget to **inject the logger** into any model that needs to log/print messages

## Command

Create a **Command Pattern** Executor and provide:

* interface ICommand
  + void Execute()
* interface IExecutor
  + void ExecuteCommand(Command command)
* Concrete Executor named CommandExecutor implements IExecutor
* Concrete Commands
  + TargetCommand with constructor (Attacker, Target)
  + AttackCommand with constructor (Attacker)

### Hints

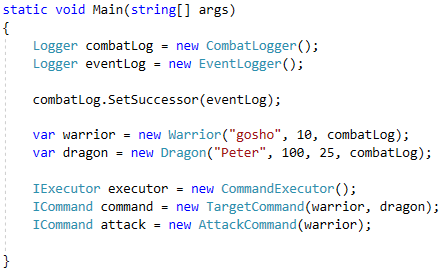
Create the interfaces

Each new command should implement ICommand, so it can be executed by the IExecutor

C:\Users\david\Documents\ShareX\Screenshots\2018-04\devenv_2018-04-05_04-11-20.png

Create as many commands as you like

Test your commands



# Part II: Mediator, Observer Design Pattern

## Mediator

Implement a Mediator Pattern groups and provide:

* interface IAttackGroup
  + void AddMember(Attacker)
  + void GroupTarget(Target)
  + void GroupAttack()
* Concrete class Group that implements IAttackGroup
* Concrete Commands:
  + GroupTargetCommand with constructor (IAttackGroup, ITarget)
  + GroupAttackCommand with constructor (IAttackGroup)

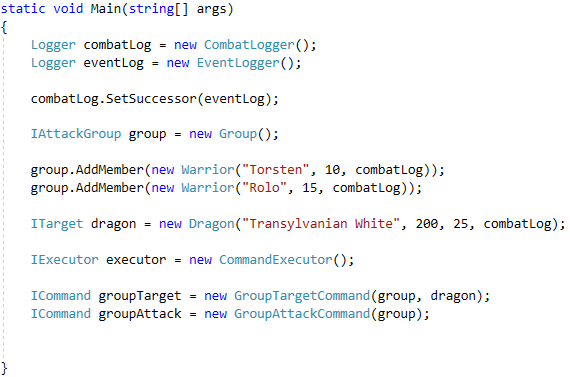
### Hints

Implement Group implements IAttackGroup - this will be the concrete mediator

x

Create some group commands, following the logic from the previous problem

Test the mediator



## Observer

Implement the **Observer Design Pattern** by providing the following:

* interface ISubject
  + void Register(Observer)
  + void Unregister(Observer)
  + void NotifyObservers()
* interface I**Observer**
  + Update(int)

If a **Target** dies, it should **send reward** to all of its **Observers**

### Hints

Create the interfaces

IAttacker should be the IObserver

\* Dragon should be the ISubject - (the easiest way is to make ITarget extends ISubject, but this is violation of the **Interface Segregation Principle**). The better solution is to create a new interface ObservableTarget and implement both ITarget and IObserver.