# Lab: Object Communication and Events

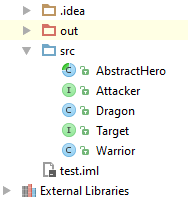
Problems for exercises and homework for the ["Java OOP Advanced" course @ SoftUni](https://softuni.bg/courses/java-oop-advanced).

You can check your solutions here: <https://judge.softuni.bg/Contests/537/Object-Communication-and-Events-Lab> .

# Part I: Chain of Responsibility, Command Design Pattern

## Resources

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



## Logger - Chain of Responsibility

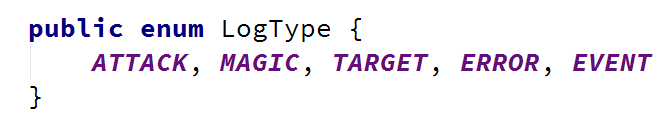
Create a **Chain of Responsibility** Logger and provide:

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

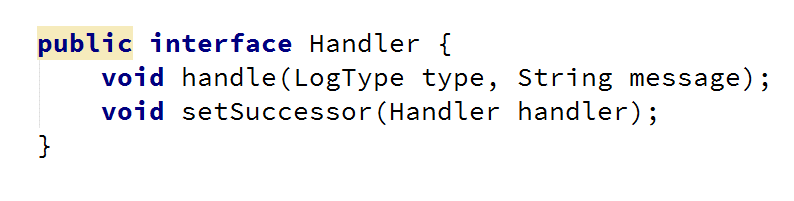
Log messages in format (**"TYPE: message"**)

### Solution

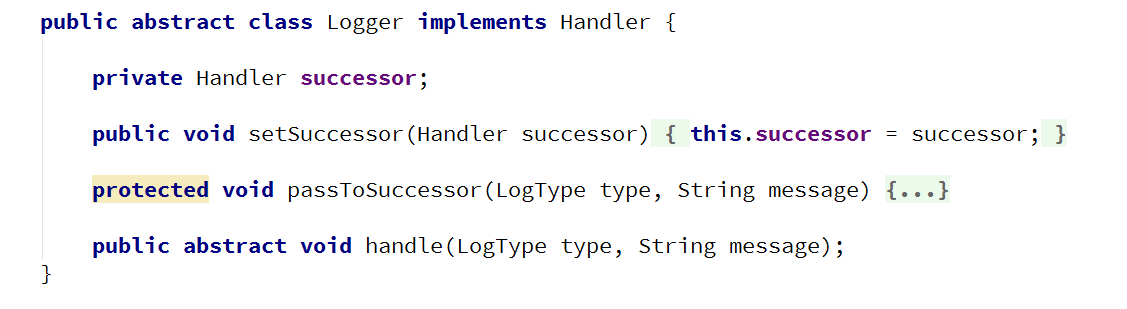
Create enum LogType



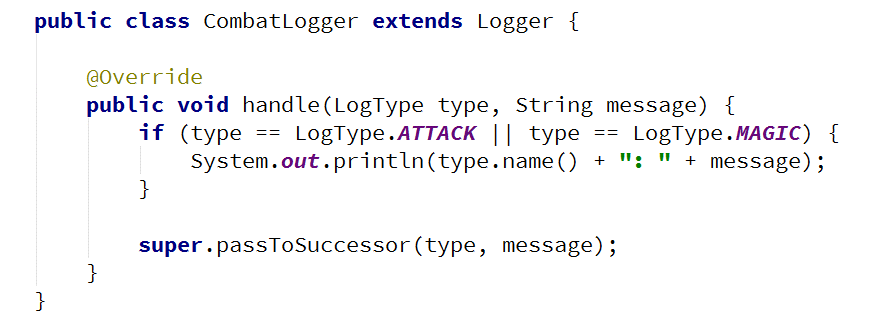
Create Handler interface



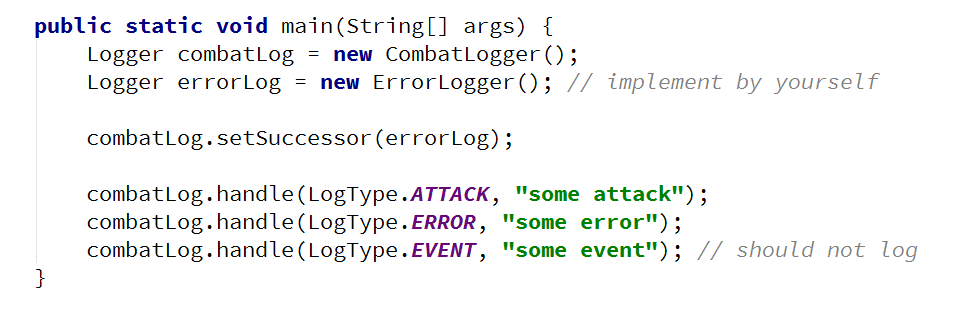
You can create an abstract logger, so you can abstract some of the functionalities



Create a concrete logger that extends Logger



Test the logger through you client



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 Command
  + void execute()
* interface Executor
  + void executeCommand(Command command)
* Concrete Executor named CommandExecutor implements Executor
* Concrete Commands
  + TargetCommand with constructor (Attacker, Target)
  + AttackCommand with constructor (Attacker)

### Hints

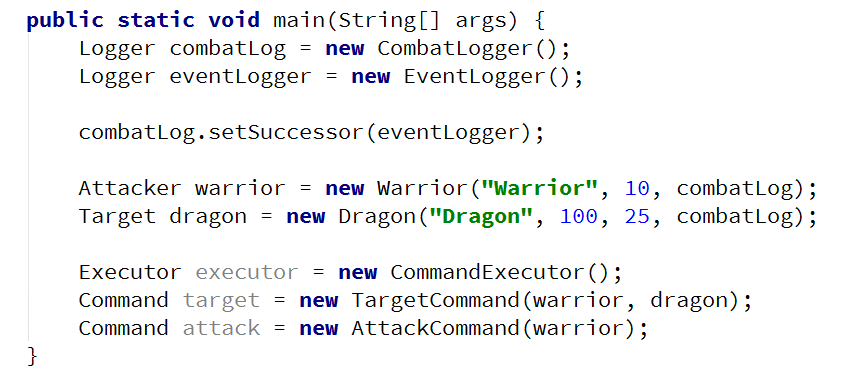
Create the interfaces

Each new command should implement Command, so it can be executed by the Executor



Create as many commands as you like

Test your commands



# Part II: Mediator, Observer Design Pattern

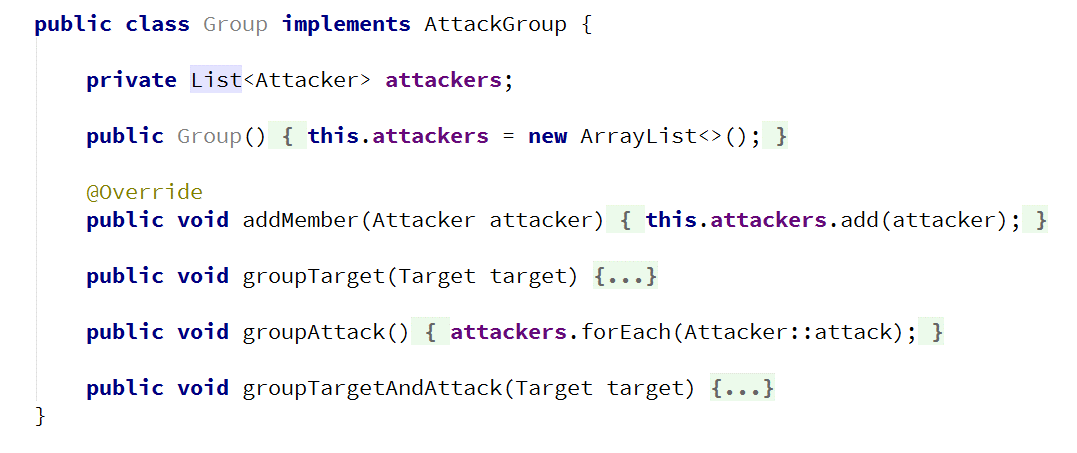
## Mediator

Implement a Mediator Pattern groups and provide:

* interface AttackGroup
  + void addMember(Attacker)
  + void groupTarget(Target)
  + void groupAttack()
* Concrete class Group that implements AttackGroup
* Concrete Commands:
  + GroupTargetCommand with constructor (AttackGroup, Target)
  + GroupAttackCommand with constructor (AttackGroup)

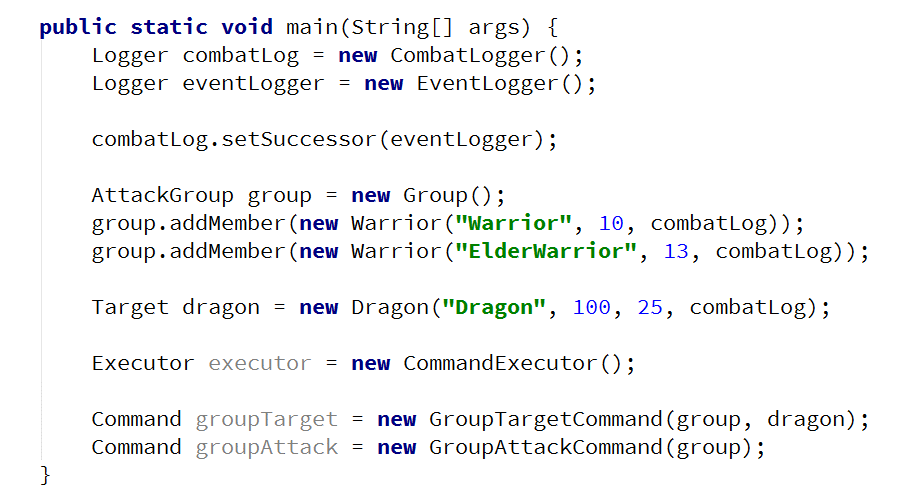
### Hints

Implement Group implements AttackGroup - this will be the concrete mediator



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 Subject
  + void register(Observer)
  + void unregister(Observer)
  + void notifyObservers()
* interface Observer
  + update(int)

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

### Hints

Create the interfaces

Attacker should be the Observer

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