# Exercises for Chapter 14 – Delegates & Events

## Exercise 1

***Delegates Exercise***

* Recall the **EmployeesManager** class, which was introduced while discussing Indexers.
* This **EmployeesManager** class provides, among other services, two methods for sorting employees: Sort by name, and Sort by Salary.
  + Both methods activate the same sorting algorithm, and they differ only in the sorting criteria.
  + These sorting services have proved to be insufficient.
  + We might want to sort by another existing **Employee** field (e.g. Id), or new fields that might be added to the **Employee** class.
    - For example, a seniority field might be added to the Employee class, and we might want to sort the employees by their seniority.
    - Currently, any such change requires an **EmployeesManager** class modification.
* We would like, however, to get the sorting service from the **EmployeesManager** class in a more general and flexible way.
  + We require that a sorting criteria change will be a “client issue” only, and will not require any modifications in the “server” class – **EmployeesManager.**
* Can you think of such a general way to define the **EmployeesManager**‘s class sort method ?
  + How should this sort method get the required sorting criteria?
  + Can it be defined independently of the sorting criteria required?
* In which class should the different ordering methods be defined?
  + What should be their prototype?
    - If you have some answers to the above questions, go ahead!   
      Enhance the **EmployeesManager** and **Employee** classes to support the new requirements. Change the **main** accordingly.
      * Otherwise, follow the instructions in the next slides.
* In the **EmployeesManager** class:
  + Define a delegate that would wrap any required ordering method between two employees.
    - The **compareCallback** delegate will accept two **Employee** objects parameters.
  + Redefine the **sort** method to get such an ordering delegate object as a parameter, and perform it on the **Employee** objects during its sorting process.
    - Keep the current simple sort algorithm. However, activate the delegate argument provided instead of a specific ordering criteria.
* In the **Employee** class:
  + Define two static ordering methods, according to the prototype of the **EmployeesManager** delegate defined above.
    - One method would provide ordering by name.
    - The other would provide ordering by salary.
  + In the **main** method, enhance the sorting calls to leverage the new delegate mechanism provided by the **EmployeesManager** class and the specific sorting methods defined in **Employee.**
* What would be the meaning of adding an Employee seniority field and require to sort by seniority?
  + Which class should be modified to support the requirement?

## Exercise 2

***A Chat System Exercise***

* Your task is to create a simple chat system!
  + The system is comprised of two classes: **ChatServer** and **ChatClient**.
  + The basic chatting scenario should be:
    - A chat client sends a message (through the ChatServer).
    - The chat server:
      * Prints the client’s name following by the message arrived.
      * Raises a **MsgArrived** event to all registered clients.
    - When a client receives a message it prints its name, followed by the arrived message and its sender.
* The **ChatServer** class should provide the following services:
  + A **SendMessage()** method, enabling chat clients to send messages.
  + The **SendMessage** method should receive the sender object and the message.
  + To enable chat clients register themselves as listeners of arrived messages, it should define the following:
    - A **MsgArrivedEventArgs** event class, holding event arguments:
      * The **ChatClient** that sent the message.
      * The sent message
  + A **MsgArrivedEventHandler** delegate type, receiving as parameters
    - The sender **object** ,and
    - A **MsgArrivedEventArgs** object.
  + Event listeners should define such a method and provide it as a callback when registering to the event.
    - An **public event** field:
    - **MsgArrivedEventHandler MsgArrived;**
* In addition, the server should define an **OnMsgArrived** method, raising the event when the message arrives.
* The **ChatClient** class should:
  + Provide users with a Chat() method that receives the chat message as parameter.
  + Define a method according to the server’s delegate type signature. This method should perform the operation required to be done when a **MsgArrived** event is notified.
  + In its constructor, the client should get a chat server object as a parameter and register to the server’s **MsgArrived** event, providing its callback method as parameter.
* Small Fix:
  + As you can see in program’s output demonstration, there is still a problem with the above chatting system.
  + The chatter that sent the message also receives it…
  + Can you think of a change that can be done to the Chat Server **OnMsgArrived()** method, such that it will notify all listening chatters except of the one that sent the message?