first part:

I think overall there is good clarity in the documents. I would have liked some more comments, and also some of the comments are in Swedish. Most of the classes, attributes and functions have a good naming standard. I'm not very fond of the naming of the classes BoatOpt and MemberOpt. I guess "Opt" is short for something but didn't manage to figure out what. The CamelCasing/PascalCasing is mostly following the C# convention. Some variables use camelCasing instead, which is little confusing when not using the same style for whole software, but very good structured code / easy readable code.

Class Diagram:
I think the class diagram misses a big part of showing the design when only consists of Member, Boat, and BoatType. I would have preferred if all classes were modeled, and good to show package so you can see that MV design is used.
It's a correct use of the composition relation line. However, the association line between Member and Boat should have an arrow to show the association, Larman shows the basic notations in a figure[1, p250]. There should also be an association relation between Boat and BoatTypes (BoatTypes class should be singular). In your design, it was modeled as a dependency. There should also be a dependency relation between Member and BoatTypes since in the code Member depends on BoatTypes. Another thing that I don't think is accurate that the Member and Boat classes variables are public, so should be changed from + to -. If they are public it should be changed to private in the code as well. Not 100% sure on the C# short way of making variables with getters and setters. It says public in front of it, but in my understanding, it's just the get and set method that is public and the variable itself is private.

Sequence Diagram:
First of all, I think the diagrams should be split into two sequence diagrams. Then call arrows are a bit messed up in the notation. Returning arrows shouldn't be filled, and all calls should have filled line as shown by Larman [1, p228-229]. But biggest error is that is not correct to the code, i.e. Member doesn't send any GetMembers message to Database. I also think that all of your own Classes that are involved should be part of each sequence diagram. So if not to start from the Program class I think you should at least start from the MemberOpt class to show the messaging to the view, returning of a statement and how the loop in MemberOpt class send messages to each Member to return the compact/verbose info.
You can also add package names in the Class names by ”::” to show what package each class belongs to. By doing so and adding the other classes, one could se that the design follows a MV architecture. Larman shows how to do this in his[1, p564]

second part:

The submission basically holds all the needed elements and not any unnecessary ones. However as mentioned before I think that the sequence diagram should be split up in two sequence diagrams.

Executable:

The system works quite fine, CRUD works pretty well for both Boat and Member. Writing to a Json file, I could find two major bugs where one would lead to an exception.
1. when I removed all boats from a member and choose Show and the id of the previous boat(was 1 in my test) I still got to the menu where I can edit, and delete boats. I chose to edit and got an Error, "Object reference not set to an instance of an object".
2. If all members are removed and I choose show member I will get to an infinite loop where the system will ask for a member ID number.

Nice look of the console view. Easy to see the options, the members and boats are nicely outputted.

third part:

Except for that the model sends view specific strings. I think that there is model-view separation. But the view specific functions should be moved to the view.
Example:

```
public string ToVerbose()
        {
            string output = $"Member {Id}: {Name} ({PersonalNumber}). Number
of boats: {numberOfBoats}";
            foreach (Boat boat in Boats)
            {
                output += "\n" + boat.ToString();
            }
            return output;
        }
```

to create the verbose list or normal list should be part of the view, and should retrieve each element from the Member instead.

There is some use of Id's instead of using references that is not very object oriented. An example of this:

```
 public void DeleteBoat(int id)
        {
            Boat boat = Boats.Find(x => x.Id == id);
            Boats.Remove(boat);
        }
```

so, you start by using chosenBoat in the BoatOpt class, but when sending the request to Member class you use an id as the argument. And after that, you iterate over the boats until you have the correct ID and assigns that boat to “boat” variable. Then you remove from the collection of boats by the reference.

Much both cleaner and faster way would be to :

```
 public void DeleteBoat(Boat boat)
        {
            Boats.Remove(boat);
```

```
            }
```
and just send in chosenBoat as the argument.
However you will need the getBoatById function you have, but I think that should be moved to view or a controller since I think that is quite specific for this UI solution. If for example, you had a GUI, you wouldn't probably need the function at all.

Another problem with the MemberOpt and ClassOpt than the naming is that they are in the base package. I think only the Program class should be there, and instead, they should be moved to "view" or to a "controller" package. But they have a little to view specific operands, so if not restructured which is maybe the best option, they should at least be moved to the view since they contain view and UI specific implementation.

In my opinion the Database class should only handling import and export to the .json file. But as implemented it holds the collection of members and is the kind of base class of the model.

So to sum it I think it's a quite good implementation that needs just slightly more work to make it a strictly MV-separation. But a little more focus should be to make the design diagrams be an accurate model of the implementation. If fixing the mentioned things, it will be a very nice submission.

I will give you a passing score, but I do that by the assumption that you fix the model-view separation and the diagrams.