# Instructions – Exercise 4.2 – Inversion of Control and Dependency Injection

composer-app, part 3

**Instructions**

* ~~Make a copy of the enhanced-composer-app from Exercise 3.2 and add it to your week-4 directory~~
* ~~Rename the application to di-composer-app~~
* ~~Delete the node\_modules directory~~
* ~~Delete the package-lock.json file~~
* ~~Open the angular.json file and find and replace all “enhanced-composer-app” entries with “di-composer-app”~~
* ~~Open the package.json file and change the name to “di-composer-app”~~
* ~~Run npm install and ng serve~~
  + ~~You are doing this to test the application and confirm there are no errors~~
* ~~Generate a new Angular service through Angular’s built-in CLI and name it composer~~
  + ~~ng g s composer~~
* composer.service.ts
  + ~~Add an import to the IComposer interface~~
    - ~~import { IComposer } from ‘./composer.interface’;~~
  + ~~Take the code from the composer.class.ts file and move it to the composer.service.ts file~~
* ~~Delete the composer.class.ts file~~
* composer-list.component.ts file
  + ~~Remove the Composer class import~~
  + ~~Add an import statement for the Composer service~~
    - ~~import { ComposerService } from ‘../composer.service.’;~~
  + ~~Add the ComposerService to the components constructor~~
    - ~~constructor(private composerService: ComposerService)~~
    - In the body of the components constructor, replace the getComposers() call from the Composer class with the composerService.getComposers() function
* composer-details.component.ts
  + ~~Remove the Composer class import~~
  + ~~Add an import statement for the Composer Service~~
    - ~~import { ComposerService } from ‘../composer.service’;~~
  + ~~Add the ComposerService to the components constructor~~
    - ~~constructor(private route: ActivatedRoute, private ComposerService: ComposerService)~~
    - In the body of the components constructor, replace the getComposer call from the Composer class with the composerService.getComposer(composerId: number) function
      * this.composerService.getComposer(this.composerId)
* ~~app.component.html~~
  + ~~Change the exercise name to Exercise 4.2 - Inversion of Control and Dependency Injection~~
* Run and test the application
  + The application should behave identical to the previous enhanced-composer-app.

Note: the goal of this exercise is to demonstrate the difference between Dependency Injection and “traditional object creation.”  In the enhanced-composer-app we created a traditional class and had to create a new instance of that class before we were able to access the getComposers() function.  In our new version, we are leveraging the built-in Dependency Injection provider from Angular to create an instance of our service.  In other words, Dependency Injection is a design pattern where the provider is responsible for creating new instances of a class.  How does Inversion of Control fit into this?  IOC is a design pattern where “some class” acts as the provider for creating new instances of a class.  Think of it as a class that creates new instances of other classes.  And, think of Dependency Injection as the name for the overall process.