**Why dependency injection?**

The main reason for using dependency injection is to

* Remove tightly coupled components by injecting dependencies rather than instantiating them directly in the consuming components.
* It has the ability to add the functionality of components at run-time.
* It is a design pattern in which a class receives its dependencies from external sources rather than creating themselves. DI allows developers to reuse the code across the application.
* DI makes the application development and testing much easier.

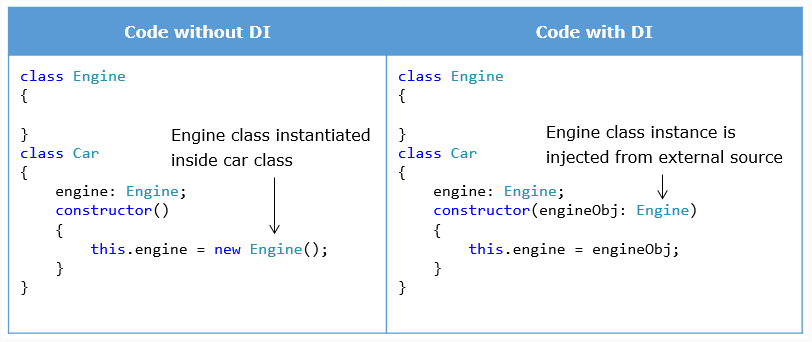
**What is Dependency injection?**

Dependency Injection (DI) is a mechanism where the required resources will be injected into the code automatically.

Angular comes with an in-built dependency injection subsystem.

**Example:**

Consider two classes Car and Engine. The relationship between them is 'a car has an engine'. In case of dependency injection, the Car class need not instantiate the Engine class. Instead, it will be passed to the Car from an external source.



**Dependency Injection in Service Creation**

You need to follow the below three steps to work with services in Angular using DI design pattern.

1. **Defining**the dependency is the creation of a service class.
2. **Registering**the dependency is registering the service class with an Angular injector.
3. **Injecting**the dependency is declaring the dependency in the components that will use the service.

Now that we have understood DI, let us move ahead to learn its implementation while creating services.

**1. Creating or defining a Service**

***book.service.ts****(unchanged)*

1. import {Book} from './book'; *// Importing the Book class type.*
2. export class BookService {
3. constructor() { }
4. getAllBooks():Book[] { *// returns an Array of types*
5. return [{"id": 1, "name": "HTML 5"},
6. {"id": 2, "name": "CSS 3"},
7. {"id": 3, "name": "Java Script"},
8. {"id": 4, "name": "Ajax Programming"}];
9. }
10. }

**2. Injecting a service**

In the component's constructor, pass the required service class as an argument. We can use this service object for our usage as shown.

1. @Component({...})
2. export class BookComponent implements OnInit {
3. bookListFromService:Book[]; *// Will store the bookList to be displayed*
4. constructor(private bookService:BookService) { } *// bookService instance passed in Component's constructor*
5. ngOnInit() {
6. this.bookListFromService = this.bookService.getAllBooks(); *// invoking the getAllBooks() of bookService to get data*
7. }
8. }

BookService will be injected into the component through constructor injection by the Angular framework.

**3. Providing for Service**

Add providers property in the module class so that service class is available to the entire application.

1. @NgModule({
2. declarations: [ AppComponent, BookComponent],
3. imports: [ BrowserModule ],
4. providers: [{
5. provide: BookService,
6. useClass: BookService
7. }],
8. bootstrap: [AppComponent]
9. })
10. export class AppModule { }

The *providers* array contains an array of provider object. Each provider object has two main properties:

* provide - the name of the service which is being requested
* useClass - the name of the service which is actually injected

Usually provide and useClass will have the same name, i.e the object requested and object injected are the same.

A ***common way*** is to just mention the Service class name (eg, BookService) in the providers' array inside the AppModule. For example, providers:[BookService]

1. @NgModule({
2. declarations: [ AppComponent, BookComponent ],
3. imports: [ BrowserModule ],
4. providers: [ BookService ],
5. bootstrap: [AppComponent]
6. })
7. export class AppModule { }

If service is needed only for the specific component(s), it can be added in those component classes instead of module.