Dependency Injection

1. **Dependency:** When class A uses some functionality of class B, then it's said that class A has a dependency of class B.
2. In most of the Object Oriented Programming languages before using methods of other classes, we need to create objects of that class.
3. **Dependency Injection:** So, transferring the task of creating the object to someone else and directly using the dependency is called dependency injection.
4. **Why do we need it?**
    a. Let's say we have a car class which contains various objects such as wheels, engines, etc, and the car class is responsible for creating all the dependent objects. Now, what if we decide to create another class of MRFWheels, Yokohama Wheels?
    b. We will need to recreate the car object with a new Yokohama dependency. But when using dependency injection (DI), we can change the Wheels at runtime (because dependencies can be injected at runtime rather than at compile time).
    c. You can think of DI as the middleman in our code who does all the work of creating the preferred wheels object and providing it to the Car class.
    d. It makes our Car class independent from creating the objects of Wheels, Battery, etc.
5. **Types of Dependencies:**
    a. **Constructor Injection:** When the Injector injects the dependency object (i.e. service) through the client class constructor, then it is called Constructor Dependency Injection.
        i. When using Constructor Injection we should check that the reference passed is not null before we proceed. Checking for null is a boilerplate code which should be avoided. Instead we can use a *guard pattern* to check for the null. The ***guard pattern*** actually is defined as any Boolean expression that must evaluate to true before the program execution can continue. It is usually used to ensure that certain preconditions are met before a method can continue, ensuring that the code that follows can properly execute.

   b. **Setter Injection:** When the Injector injects the dependency object (i.e. service) through the public property of the client class, then it is called Setter Dependency Injection. This is also called the Property Injection.
   c. **Method Injection:** When the Injector injects the dependency object (i.e. service) through a public method of the client class, then it is called Method Dependency Injection.
6. By using Dependency Injection, it's the dependency Injection responsibility:
   a. Create the objects
   b. Know which classes require those objects
   c. And provide them all those object
7. If there is any change in objects, then DI looks into it and it should not concern the class using those objects. This way if the object's change in the future, then it's DI's responsibility to provide the appropriate objects to the class.
8. **Inversion of Control(IoC) - Concept behind the DI:** This states that a class should not configure its dependencies statically but should be configured by some other class from outside.
   a. It is the fifth principle of **S.O.L.I.D** which states that a class should depend on abstraction and not upon concretions (in simple terms, hard-coded).
   b. According to the principles, a class should concentrate on fulfilling its responsibilities and not on creating objects that it requires to fulfil those responsibilities. And that's where **Dependency Injection** comes into play: it provides the class with the required objects.
9. **Benefits of DI:**
   a. Helps in Unit testing.
   b. Boilerplate code is reduced, as initialising of dependencies is done by the injector component.
   c. Extending the application becomes easier.
   d. Helps to enable loose coupling, which is important in application programming. DI is used to reduce the tight coupling between the software components. As a result, we can easily manage future changes and other complexity in our application.

10. **Disadvantages of DI:**
    a. It's a bit complex to learn, and if overused can lead to management issues and other problems.
    b. Many compile time errors are pushed to run-time.
    c. Dependency injection frameworks are implemented with reflection or dynamic programming. This can hinder use of IDE automation, such as "find references", "show call hierarchy" and safe refactoring.
11. **Libraries and Frameworks that implement DI:**
    a. Spring (Java)
    b. Google Guice (Java)
    c. Dagger (Java and Android)
    d. Castle Windsor (.NET)
    e. Unity(.NET)
12. **Tight Coupling:** Tight coupling means two objects are dependent on each other. That means when a class is dependent on another class, then it is said to be a tight coupling between these two classes. In that case, if we change the dependent object, then we also need to change the classes where this dependent object is used. If your application is a small one, then it is not that difficult to handle but if you have a big enterprise-level application, then it's really very difficult to handle making these changes.
13. **Dependency Injection pattern involves 3 types of classes:**
    1. **Client Class:** The Client class (dependent class) is a class that depends on the service class.
    2. **Service Class:** The Service class (dependency) is a class that provides service to the client class.
    3. **Injector Class:** The Injector class injects the service class object into the client class.
    4. **Note:** If we use Interfaces, then Unit Testing will become easy.
14. **Sample Code:**
    https://github.com/Yashwanth-G/Dependency_Injection_Implementation
15. **Usage:**
    a. **When to use Constructor Injection:** You should use constructor injection when your class has a dependency that the class requires in order to work properly. If your class cannot work without a dependency, then inject it via the constructor. If your class needs three dependencies, then demand all three in the constructor.

Additionally, you should use constructor injection when the dependency in question has a lifetime longer than a single method.

b. **When to use Setter/Property Injection:** Use property injection when a dependency is optional and/or when a dependency can be changed after the class is instantiated. Use it when you want users of the containing class to be able to provide their own implementation of the interface in question.
c. **When to use Method Injection:** Method injection should be used when the dependency could change with every use, or at least when you can't be sure which dependency will be needed at the point of use.

Ref:
https://betterprogramming.pub/the-3-types-of-dependency-injection-141b40d2cebc

Ref:
https://www.freecodecamp.org/news/a-quick-intro-to-dependency-injection-what-it-is-and-when-to-use-it-7578c84fa88f/

***** THIS DOCUMENT IS PREPARED FOR MY SELF PREPARATION ONLY. *****