Dependency Injection with Real-Life Examples

Dependency Injection (DI) is a design pattern used in software engineering to reduce tight coupling between components, making code more modular, testable, and maintainable. It allows objects to receive their dependencies from external sources rather than creating them internally.

🛠️ Types of Dependency Injection

1. **Constructor Injection** Dependencies are passed via the class constructor.

`python class Controller: def __init__(self, repository): self.repo = repository`

1. **Setter Injection** Dependencies are set via setter methods after object construction.

`python controller = Controller() controller.set_repository(repo)`

1. **Interface Injection** (less common in Python) The dependency exposes a method that the consumer calls to pass itself.

2. Testing with Mocks

```
class MockService:
    def process(self):
        print("Mocked!")

client = Client(MockService())
client.do_work()  # prints "Mocked!"
```

DI makes testing easy without needing complex setups or modifying core logic.

4. Framework Support

- **Spring (Java)** – built-in support for DI using annotations like `@Autowired`
- **Angular** – powerful DI system for injecting services into components
- **FastAPI / Flask / Django** – can implement DI via constructor-based or third-party libs like `dependency-injector`

⚠️ Caveats

- Can introduce complexity
- Overhead of managing containers/injectors
- May reduce readability for newcomers

🔚 Conclusion

Dependency Injection fosters flexibility and testability in modern codebases. While not always necessary in simple scripts, it becomes indispensable in large-scale applications and microservices where decoupling is key.

Use it when:

- Components have multiple responsibilities
- You need testability with mocks/stubs
- You want to follow SOLID principles (esp. the D in **D**IP)