Козлов А.В.
Научный руководитель: канд. техн. наук, доцент А. А. Белов
Муромский институт (филиал) федерального государственного образовательного учреждения высшего образования «Владимирский государственный университет имени Александра Григорьевича и Николая Григорьевича Столетовых»
602264, г. Муром, Владимирская обл., ул. Орловская, 23
e-mail: kaf-eivt@yandex.ru

Архитектурный паттерн «Dependency Injection»

В программировании, когда класс A использует класс или интерфейс B, тогда A зависит от B. A не может выполнить свою работу без B, и A не может быть переиспользован без переиспользования B. В таком случае класс A называют «зависимым», а класс или интерфейс B называют «зависимостью».

Два класса, которые используют друг друга, называют связанными. Связанность между классами может быть или слабой, или сильной, или чем-то средним. Сильная связанность ведет к сильным зависимостям, и слабая связность ведет к слабым зависимостям или даже к отсутствию зависимостей в некоторых ситуациях. Зависимости, или связи имеют направленность. То, что A зависит от B не значит, что B зависит от A.

Недостатком зависимостей является то, что они снижают переиспользование, что негативно по многим причинам. Обычно переиспользование оказывает позитивное влияние на скорость разработки, качество кода, читаемость кода и т.д.

В ходе разработки программных приложений на высокоуровневых языках программирования, можно использовать архитектурный паттерн «Внедрение зависимостей» Dependency Injection (DI).

Внедрение зависимостей — это стиль настройки объекта, при котором поля объекта задаются внешней сущностью. Другими словами, объекты настраиваются внешними объектами. DI — это альтернатива самонастройке объектов.

Приведем пример класса MessageService, в котором не используется DI:

```
public class MessageService {
   private DataSource dataSource = new DataSourceImpl(
    "driver", "url", "user", "password"
   );
   public Message findOneById(int id) {...}
}
```

Этот класс нуждается в экземпляре DataSource для того, чтобы получить подключения к базе данных. Подключения к БД используются для чтения и записи в БД, например, объектов Person.

Класс MessageService создает экземпляр DataSourceImpl, так как нуждается в источнике данных. Тот факт, что MessageService нуждается в реализации DataSource, означает, что он зависит от него. Он не может выполнить свою работу без реализации DataSource. Следовательно, MessageService имеет «зависимость» от интерфейса DataSource и от какой-то его реализации.

В том случае, когда класс разрешает собственные зависимости, он становится негибким в отношении к этим зависимостям. Это значит, что если вам нужно поменять зависимости, вам нужно поменять код. В данном примере это означает, что если вам нужно использовать другую базу данных, вам потребуется поменять класс MessageService. Если у вас много классов доступа к данным Data Access Object (DAO), реализованных таким образом, вам придется изменять их все. Вдобавок, вы не можете провести юнит-тестирование класса, подменив реализацию DataSource. Вы можете использовать только DataSourceImpl.

Класс MessageSerivce может быть более независимым. Сейчас он все еще зависит от класса DataSourceImpl. Нет необходимости зависеть от чего-то, кроме интерфейса DataSource. Это может быть достигнуто введением DataSource в конструктор:

```
public class MessageService {
   public MessageService(DataSource dataSource) {
      this.dataSource = dataSource;
   }

   public Message findOneById(int id) {...}
}
```

Теперь класс MessageService больше не зависит от класса DataSourceImpl и можно использовать любую реализацию DataSource в конструкторе MessageService.

Паттерн «Dependency Injection» должен продолжаться через все слои приложения, с самого нижнего слоя (слоя доступа к данным) до пользовательского интерфейса (если он есть).