Patterns

Есть 3 понятия:

- Inversion of Control (IoC)
- Dependency Injection (DI)
- Dependency Inversion Principle (DIP)

По материалам:

- Martin Fowler - InversionOfControl
- Martin Fowler - Inversion of Control Containers and the Dependency Injection pattern
- Martin Fowler - DIP in the wild
- Sergey Teplyakov - DI vs. DIP vs. IoC

Inversion of Control (IoC)

Wikipedia, Stackoverflow

- Общее понятие, которое отличает библиотеку от фреймворка
- Есть традиционное процедурное программирование, когда мы определяем время, порядок вызовов библиотченых методов
- А есть framework и возможности его расширения. Фреймворк вызывает части нашего кода, которые получают управление от него

Dependency Injection (DI)

Wikipedia - DI

- Механизм передачи классу его зависимостей
- Существует несколько конкретных видов или паттернов внедрения зависимостей:
    - внедрение зависимости через конструктор (Constructor Injection),
    - через метод (Method Injection),
    - через свойство (Property Injection).

```
class ReportProcessor
{
  private readonly IReportSender _reportSender;

  // Constuctor Injection: передача обязательной зависимости
  public ReportProcessor(IReportSender reportSender)
  {
    _reportSender = reportSender;
    Logger = LogManager.DefaultLogger;
  }
  // Method Injection: передача обязательных зависимостей метода
  public void SendReport(Report report, IReportFormatter formatter)
  {
    Logger.Info("Sending report...");
    var formattedReport = formatter.Format(report);
    _reportSender.SendReport(formattedReport);
    Logger.Info("Report has been sent");
  }
  // Property Injection: установка необязательных "инфраструктурных" зависимостей
  public ILogger Logger { get; set; }
}
```

Dependency Inversion Principle (DIP)

Wikipedia - DIP

- Принцип инверсии зависимости говорит о том, к каким видам зависимостей нужно стремиться.
- Важно, чтобы зависимости класса были понятны и важны вызывающему коду.
- Зависимости класса должны располагаться на текущем или более высоком уровне абстракции

```
class ReportProcessor
{
    private readonly ISocket _socket;
    public ReportProcessor(ISocket socket)
    {
        _socket = socket;
    }
    public void SendReport(Report report, IStringBuilder stringBuilder)
    {
        stringBuilder.AppendFormat(CreateHeader(report));
        stringBuilder.AppendFormat(CreateBody(report));
        stringBuilder.AppendFormat(CreateFooter(report));
        _socket.Connect();
        _socket.Send(ConvertToByteArray(stringBuilder));
    }
}
```