„1. Single Responsibility Principle (SRP)“

Das **Single Responsibility Principle (SRP)** oder auf Deutsch das **Prinzip der einzigen Zuständigkeit** ist eines der grundlegendsten Prinzipien in der objektorientierten Programmierung und gehört zu den SOLID-Prinzipien. Es besagt, dass **eine Klasse oder ein Modul nur eine einzige Aufgabe haben sollte** oder nur **einen Grund zur Änderung** haben darf.

- In diesem Zusammenhang bedeutet es, dass eine Klasse oder ein Modul nur für **eine bestimmte Funktion** oder **eine spezifische Gruppe von Aufgaben** verantwortlich sein sollte. Wenn sich eine Klasse mit zu vielen verschiedenen Verantwortlichkeiten befasst, wird der Code schwerer zu verstehen, zu testen und zu warten.

Beispiel für das SRP:

- Ohne SRP:

```
public class EmployeeService
{
    public void EmployeeRegistration(Employee employee)
    {
        Employees.Add(employee);
        SendEmail(employee.Email, "Registration", "Congratulation !");
    }
    private void SendEmail(string email, string subject, string message)
    {
        var emailMessage = new MimeMessage();
        emailMessage.From.Add(new MailboxAddress("Mark Adam", "madam@sample.com"));
        emailMessage.To.Add(new MailboxAddress(string.Empty, email));
        emailMessage.Subject = subject;
        emailMessage.Body = new TextPart("plain") { Text = message };

        using (SmtpClient smtpClient = new SmtpClient())
        {
            smtpClient.LocalDomain = "sample.com";
            smtpClient.Connect("smtp.relay.uri", 25, SecureSocketOptions.None);
            smtpClient.Send(emailMessage);
            smtpClient.Disconnect(true);
        }
    }
}
```

- Mit SRP_Prinzip:

```
public class EmployeeService
{
  public void EmployeeRegistration(Employee employee)

    {
        Employees.Add(employee);

        EmailService.Send(
          employee.Email,
          "Registration", "Congratulation!“
        );
    }

}
```

```
public class EmailService {

 private void Send(string email,string subject,string message)
 {
  var emailMsg = new MimeMessage();
      emailMsg.From.Add(new MailboxAddress(“Name”, email));
      emailMsg.To.Add(new MailboxAddress(“Name”, email));
      emailMsg.Subject = subject;
      emailMsg.Body = new TextPart("plain") {Text=message};

   using (SmtpClient smtpClient = new SmtpClient())
    {
        smtpClient.LocalDomain = "sample.com";
        smtpClient.Connect("smtp",25,SecureSocketOptions.None);
        smtpClient.Send(emailMessage);
        smtpClient.Disconnect(true);
    }
  }
}
```

Beispiel2:

- Ohne SRP:
Das verstößt aber gegen das **Single Responsibility Principle**, weil die Klasse `Rechnung` mehrere Gründe zur Änderung hat:

java — Code kopieren

```java
class Rechnung {
    public void berechneRechnungsbetrag() {
        // Logik zur Berechnung des Rechnungsbetrags
    }

    public void druckeRechnung() {
        // Logik zum Drucken der Rechnung
    }

    public void speichereRechnungInDatenbank() {
        // Logik zum Speichern der Rechnung in der Datenbank
    }
}
```

- nach dem Single Responsibility Principle:

java — Code kopieren

```java
class Rechnungsberechnung {
    public void berechneRechnungsbetrag() {
        // Logik zur Berechnung des Rechnungsbetrags
    }
}

class Rechnungsdrucker {
    public void druckeRechnung() {
        // Logik zum Drucken der Rechnung
    }
}

class Rechnungsdatenbank {
    public void speichereRechnung() {
        // Logik zum Speichern der Rechnung in der Datenbank
    }
}
```

Vorteile dieser Struktur:

- **Einfachere Wartung:** Wenn sich die Art und Weise der Rechnungsspeicherung ändert, müssen wir nur die Klasse `Rechnungsdatenbank` anpassen, ohne den Rest des Codes zu berühren.
- **Bessere Testbarkeit:** Wir können jede Klasse isoliert testen, was die Tests fokussierter und schneller macht.
- **Klarere Verantwortlichkeiten:** Es ist einfacher zu erkennen, welche Klasse für welche Funktion verantwortlich ist, was den Code lesbarer und verständlicher macht.