31.8.–3.9.2015
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Trypanophobia

Keine Angst vor Injektionen mit Java 8

Robert Bräutigam

MATHEMA Software GmbH

Róbert Bräutigam, MATHEMA Software GmbH.

Trypanophobia

Keine Angst vor Injektionen mit Java 8

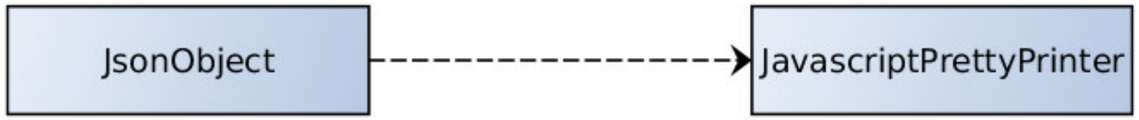
Was sind Abhängigkeiten (Dependencies)?

Die Funktionisweise einer Komponente hängt von eine andere ab.

```
1  public class JsonObject {
2      ...
3
4      @Override
5      public String toString() {
6          return new JavascriptPrettyPrinter().print(createString()); // Abhängig
7      }
8  }
```

```
1  /**
2   * Formats any JavaScript code to be human readable,
3   * indented, and ANSI colored.
4   */
5  public class JavascriptPrettyPrinter {
6      ...
7
8      public String print(String data) {
9          ...
10     }
11 }
```

Nachteile




```
1  public class JsonObject {
2      ...
3
4      @Override
5      public String toString() {
6          return new JavascriptPrettyPrinter().print(createString()); // Abhängig
7      }
8  }
```

- Defaultverhalten in toString() nicht änderbar!
- Die toString() Methode ist nicht testbar!
- **Semantisch falsch**

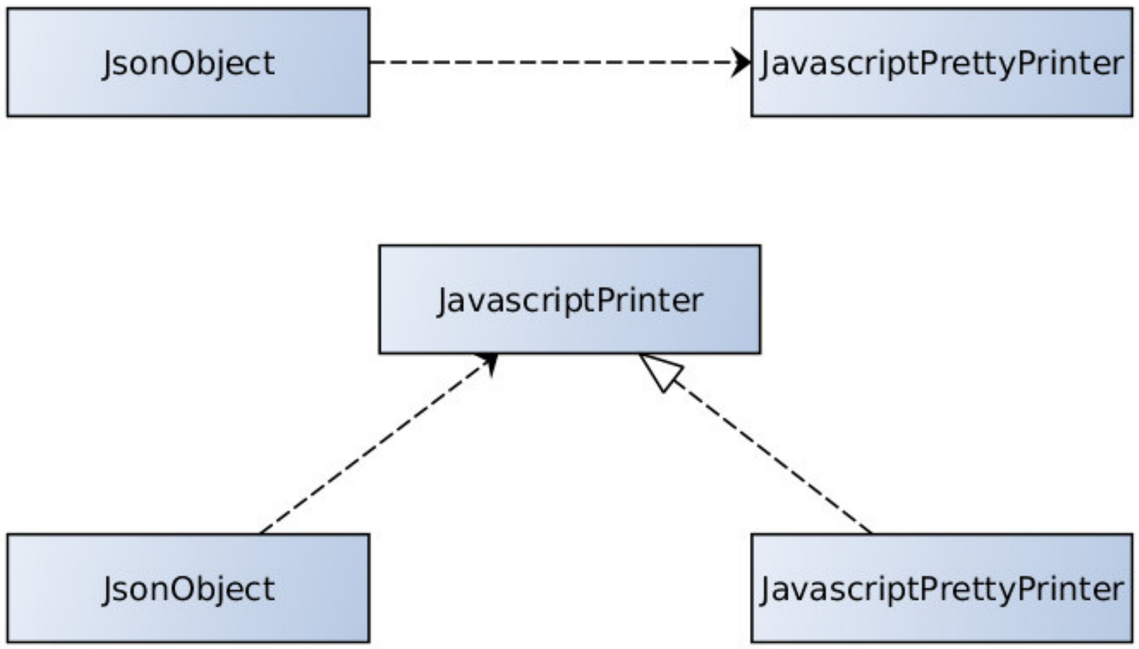
Die Lösung

```java
public class JsonObject {
   private JavascriptPrinter printer;
   ...

   @Override
   public String toString() {
      return printer.print(createString()); // Keine abhängigkeit auf Javascr
   }
}
```

```java
public interface JavascriptPrinter {
   String print(String javascript);
}
```

Die Lösung


JsonObject
JavascriptPrettyPrinter
JavascriptPrinter
JsonObject
JavascriptPrettyPrinter

Problemstellung

```
1  public class JsonObject {
2      private JavascriptPrinter printer;
3      ...
4  }
```

- Wie wird das JavascriptPrinter übergeben?

Einfache OO Lösung: Konstruktor DI

```
1  public class JsonObject {
2     private JavascriptPrinter printer;
3
4     public JsonObject(JavascriptPrinter printer) {
5        this.printer = printer;
6     }
7
8     ...
9  }
```

Weil:

- Das Printer Objeckt **muss** immer angegeben sein.
- OO Grundlage: nach Konstruktor Aufruf muss ein Objekt in einen gültigen Zustand sein.

Was fehlt noch?

```
1  public class JsonObject {
2     private JavascriptPrinter printer;
3
4     public JsonObject(JavascriptPrinter printer) {
5        this.printer = printer;
6     }
7
8     ...
9  }
```

1. Was ist wenn immer neue Objekte gebraucht werden? Immer ein Factory zu implementieren ist umständlich.
2. Was ist mit singleton Objekte/Services?
3. Was ist wenn das Objekt immer von Kontext abhängt? Beispielsweise: Transaktion, Session, usw.
4. Was ist mit Teilbäume in Libraries (Modularisation)?

Pure Java, No Magic!

- keine Änderungen an OO Code (auch Annotationen nicht!)
- Compile-Zeit Sicherheit
- typsicher / explizit
- keine externe Konfiguration, nur Java Code
- kein Classpath-Scanning
- kein Reflection
- keine Bytecode Manipulation
- keine Proxies
- kein Code Genration

Implement

Fazit

DI einfach, explizit und sicher ist möglich mit Java 8, minimales Framework reicht

```
1  public interface MyLayerModule extends StandardScopesSupport {
2     default UserRepository getUserRepository() {
3        return singleton( () -> new DatabaseUserRepository(getDataSource()) );
4     }
5
6     default DataSource getDataSource() {
7        return singleton( () -> ... );
8     }
9     ...
10 }
```

Fragen?

Danke!

robert@mathema.de