9º Meetup

Grude4J

26/09/2019

Desmistificando o IoC Container do Spring Boot

Quem usa o Spring Boot costuma dizer que ele faz "mágica", sem configuração alguma praticamente tudo funciona. Vamos entender como esse framework usando Inversion of Control, Dependecy Injection, Annotations, Reflection e outras coisas faz isso acontecer!

Flaviano Flauber

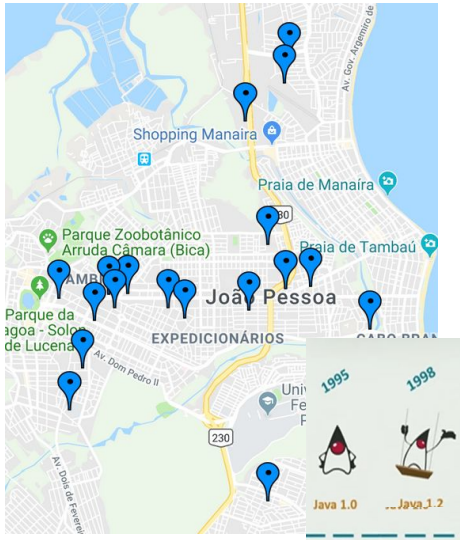
Full Stack Engineer

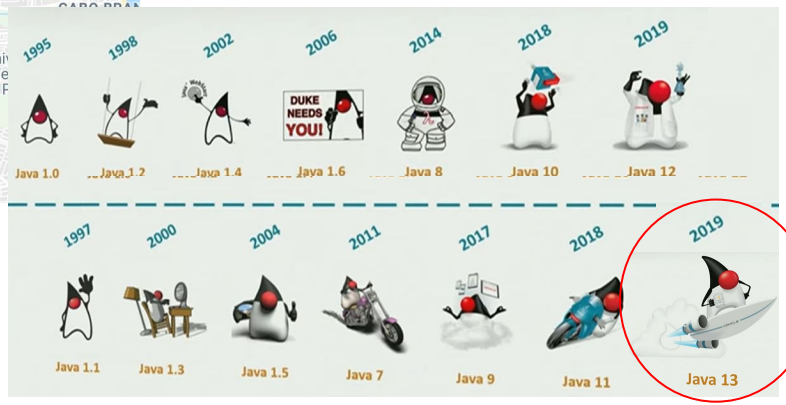
http://bit.ly/flauber

Sobre

Flaviano Flauber é full stack engineer na PBSoft, alocado no TCE-PB, desenvolvendo sistemas usando principalmente **Angular2+** e Java **Spring Boot**. Membro engajado de várias comunidades, ajuda a organizar os meetups da comunidade Grude4J. É de família, gosta de vôlei, bike e games.

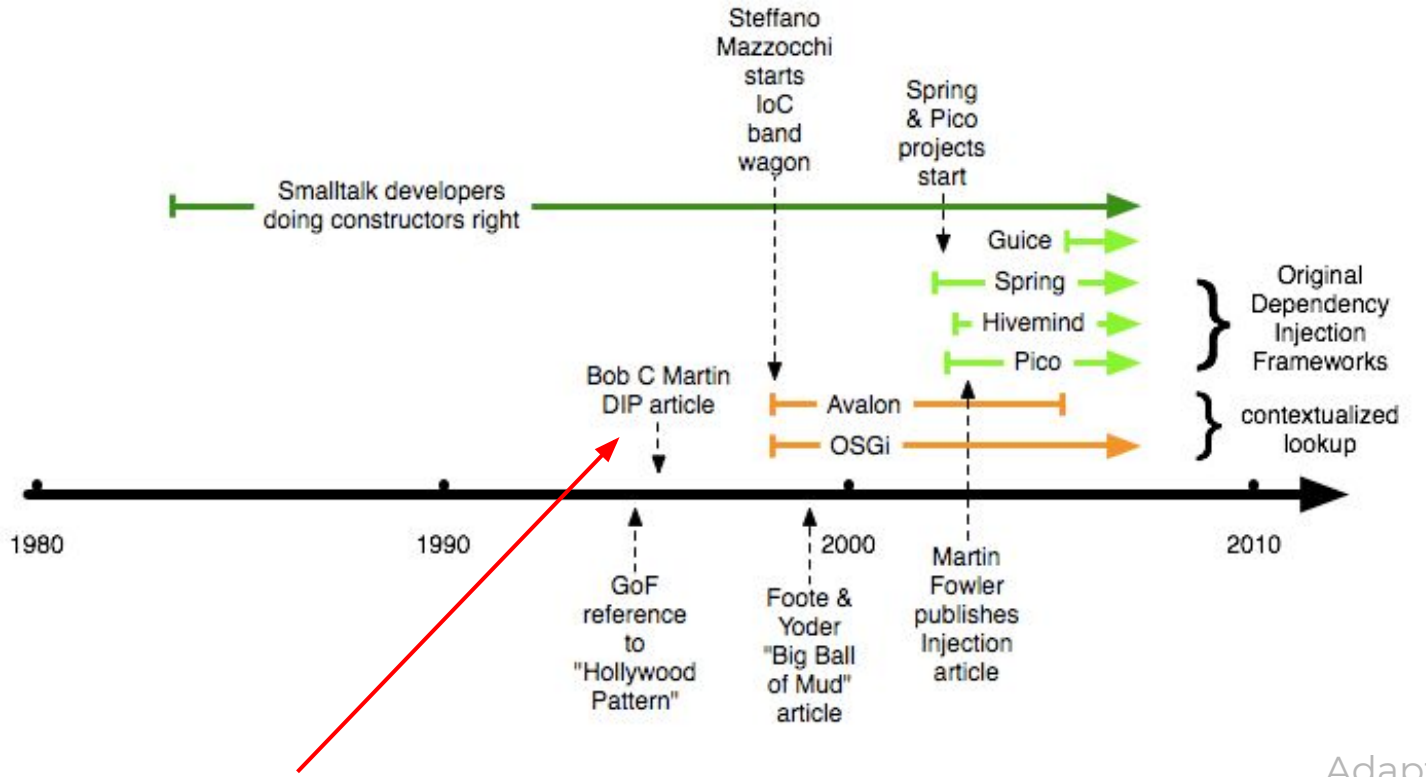
Está sempre de olho na alta empregabilidade





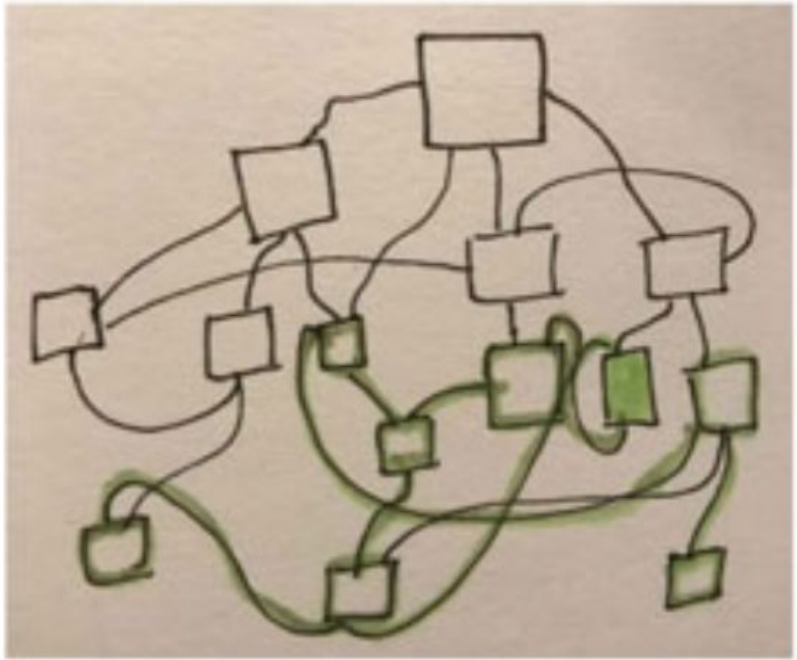
Injeção de Dependência

História



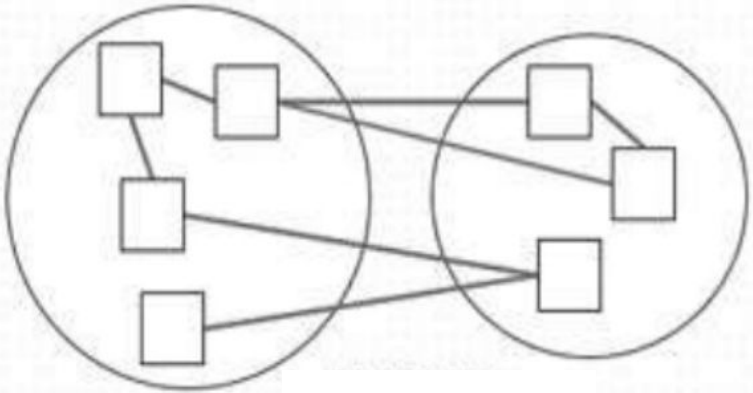
Adaptado de [3]

Big Ball of Mud

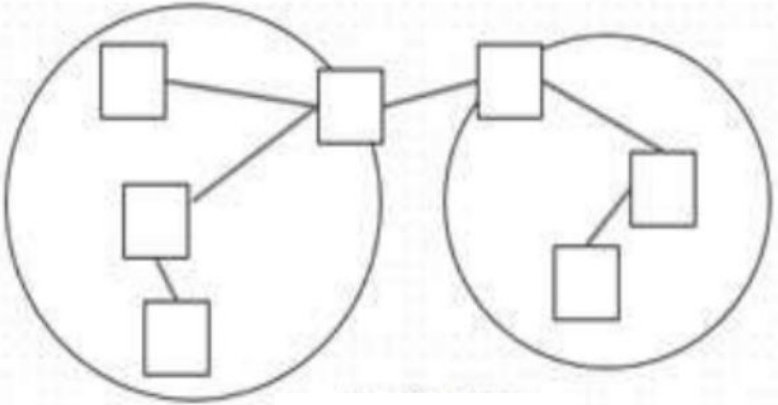


"You reach for the banana, and get the entire gorilla"
– Michael Stahl

Acoplamento e Coesão



Alto
Acoplamento



Baixo
Acoplamento

Adaptado de [1]

SOLID

SRP - Princípio da Responsabilidade Única

Sem SRP

```
1  public class DebitoContaCorrente
2  {
3      public void ValidarSaldo(int valor) { }
4
5      public void DebitarConta(int valor) { }
6
7      public void EmitirComprovante() { }
8  }
```

Considerando SRP

```
1  public class DebitoContaCorrente
2  {
3      public void DebitarConta(int valor) { }
4  }
5
6  public class SaldoContaCorrente
7  {
8      public void ValidarSaldo(int valor) { }
9  }
10
11 public class ComprovanteContaCorrente
12 {
13     public void EmitirComprovante() { }
14 }
```

Alguns benefícios do uso desse princípio:

- Complexidade do código reduzida, mais explícita e direta;
- Facilitação da legibilidade;
- Redução de acoplamento;
- Código limpo e testável;
- Facilidade de evolução.

Adaptado de [2]

SOLI*D*

DIP - Princípio da inversão da dependência

Dependency injection for five-year-olds

When you go and get things out of the refrigerator for yourself, you can cause problems. You might leave the door open, you might get something Mommy or Daddy doesn't want you to have. You might even be looking for something we don't even have or which has expired.

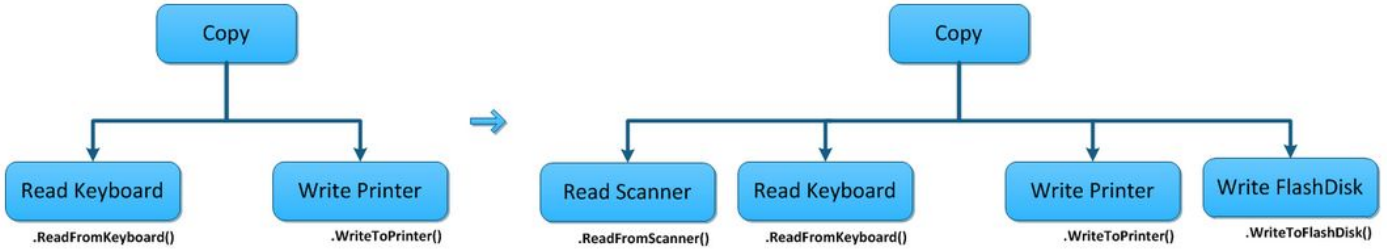
What you should be doing is stating a need, "I need something to drink with lunch," and then we will make sure you have something when you sit down to eat.
John Munsch, 28 October 2009.

Adaptado de [7]

SOLID

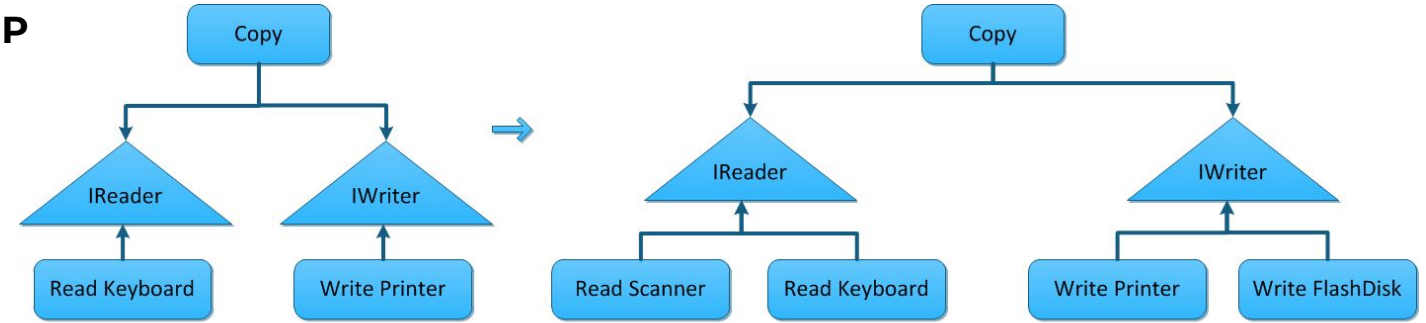
DIP - Princípio da inversão da dependência

Sem DIP



(Figure – 1.a) (Figure – 1.b)

Com DIP



(Figure – 2.a) (Figure – 2.b)

Adaptado de [5]

Java Reflection API

Inspecionando métodos de uma classe com Reflection:

```
Method[] methods = MyObject.class.getMethods();

for(Method method : methods){
    System.out.println("method = " + method.getName());
}
```

Java Annotation API

Criando/Definindo uma annotation:

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)

public @interface MyAnnotation {
    public String name();
    public String value();
}
```

Exemplo de uso:

```
@MyAnnotation(name="someName",  value = "Hello World")
public class TheClass {
}
```

Adaptado de [4]

IoC Container do Spring Boot

Spring Boot is an open source Java-based framework used to create a micro Service.

Spring IoC Container is a process whereby objects define their dependencies (that is, the other objects they work with) only through constructor arguments, arguments to a factory method, or properties that are set on the object instance after it is constructed...

Show me the code #1, Spring Boot em ação

Steps:

1. Criar uma spring boot app com a dependência web;
2. Criamos uma interface que define um único método que retorna um nome;
3. Implementamos duas classes que implementam a interface;
4. Criar um Controller que tem uma dependência da interface;
5. Criamos um endpoint que retorna um nome;
6. Alternamos entre as classes que implementam a interface usando configurando via annotation;
7. O endpoint retorna o nome em função do objeto inserido.

Show me the code #2, injetando uma dependência via annotations e reflection

Steps:

1. A classe principal do nosso programa Java possui dependência de uma interface;
2. Carregamos classes dinamicamente (no demo é pelo nome da classe);
3. Processamos as classes via reflection. Consideraremos as que implementam a interface esperada pleo programa principal. Escolhemos a que possui a annotation @MyAnnotation;
4. Instanciamos o objeto e injetamos na classe principal;
5. Imprimimos o nome em função do objeto inserido.

Adaptado de https://goo.gl/E2aV97

Referências

1. Coupling and Cohesion in Software Engineering. URL: < https://www.slideshare.net/AdilAslam4/coupling-and-cohesion-in-software-engineering >. Acessado em: 24/09/2019;
2. SOLID – Single Responsibility Principle – SRP. URL: < https://www.eduardopires.net.br/2013/05/single-responsibility-principle-srp/ >. Acessado em: 24/09/2019;
3. Dependency injection. URL: < http://memex.cc/Dependency_injection >. Acessado em: 24/09/2019;
4. Java Reflection. URL < http://tutorials.jenkov.com/java-reflection/index.html >. Acessado em: 26/09/2019;
5. Dependency Inversion Principle, IoC Container, and Dependency Injection. URL < https://www.codeproject.com/Articles/465173/Dependency-Inversion-Principle-IoC-Container-Depen >. Acessado em: 26/09/2019;
6. Software Architecture and Related Concerns. URL: < http://www.bredemeyer.com/whatis.htm >. Acessado em: 26/09/2019;
7. Dependency injection. URL: < https://en.wikipedia.org/wiki/Dependency_injection >. Acessado em: 26/09/2019;
8. Introduction to the Spring IoC Container and Beans. URL: < https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#beans-introduction >. Acessado em: 26/09/2019;