Google Guice

Framework de Injeção de Dependência

Equipe: Jozimar Soares da Costa
Lyndemberg Batista Nery
Rômulo Soares Bezerra

https://github.com/codeandcry

INVERSÃO DE CONTROLE

Inversão de Controle ou Inversion of Control mais conhecido pela sigla IoC é um padrão que busca fazer com que o controle das instâncias de uma determinada classe seja tratado externamente e não dentro da classe em questão.

Seu objetivo é inverter o controle de uma classe delegando o mesmo para uma outra classe, interface, componente, serviço, etc.

INVERSÃO DE CONTROLE

Sem usar inversão de controle

CLIENTE

SERVIÇO

INVERSÃO DE CONTROLE

Usando IoC

CLIENTE

SERVIÇO

INJEÇÃO DE DEPENDÊNCIA

Injeção de dependência (Depedency Injection ou apenas DI) é um design pattern utilizado para manter o acoplamento fraco entre classes ou módulos do sistema.

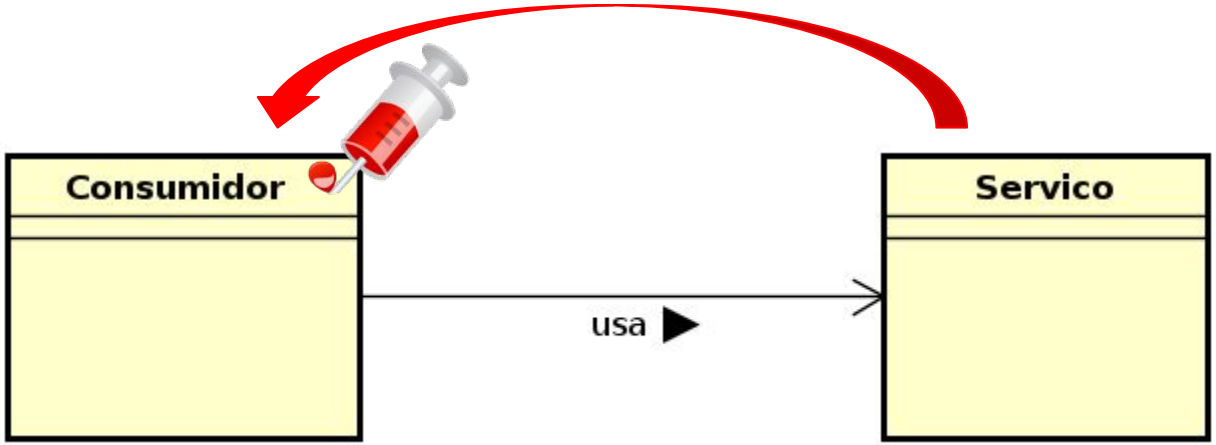
O foco principal é fazer com que uma classe não tenha conhecimento de como instanciar um objeto de um tipo do qual é dependente

INJEÇÃO DE DEPENDÊNCIA

Mas quando uma classe é dependente de um objeto?

Imagine a classe B como variável de instância da classe A. Quando você instanciar a classe A terá que ter um objeto instanciado da classe B também, caso não tenha, poderá ter uma exceção do tipo nullpointer e também não terá acesso aos métodos de B.

INJEÇÃO DE DEPENDÊNCIA



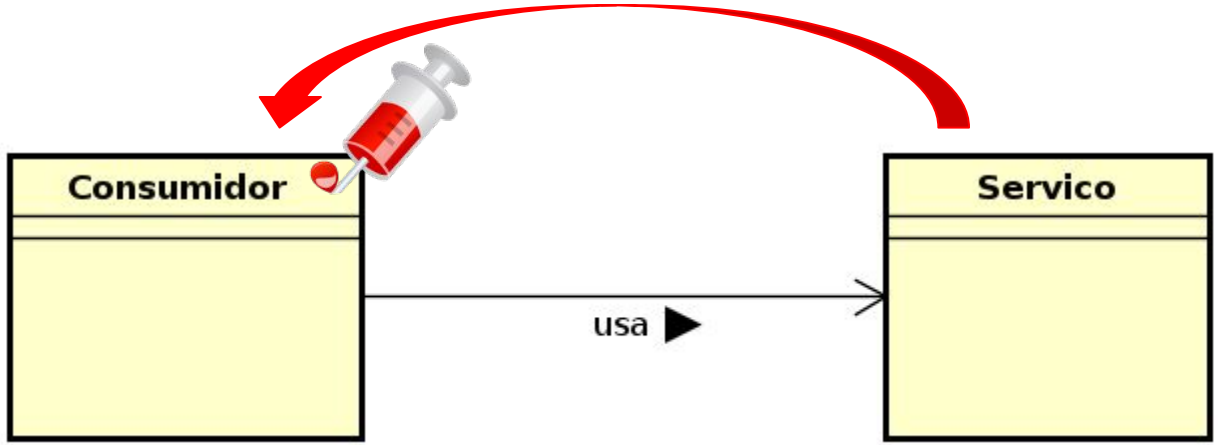
Dependente

Dependência

INJEÇÃO DE DEPENDÊNCIA

Atinge injeção de dependência

Não atende inversão de controle



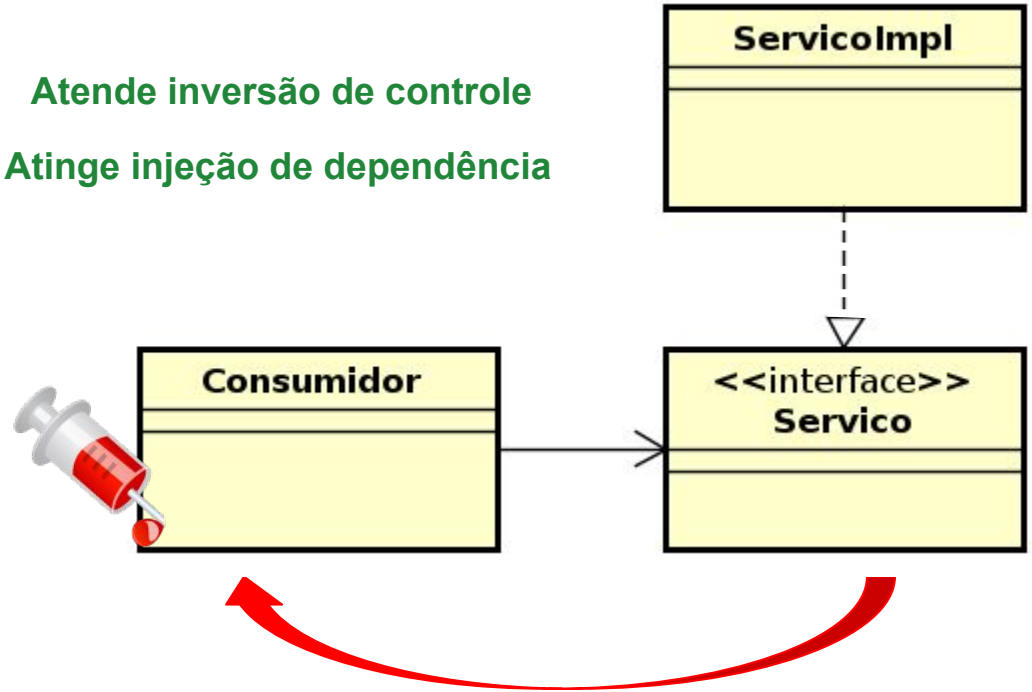
Dependente

Dependência

INJEÇÃO DE DEPENDÊNCIA

Atende inversão de controle

Atinge injeção de dependência



INJEÇÃO DE DEPENDÊNCIA

Vantagens:

- ➔ Desacoplamento.
- ➔ Reusabilidade.
- ➔ Menor custo de manutenção.
- ➔ Códigos mais limpos.

GOOGLE GUICE

Guice é um projeto OpenSource interno da Google que foi desenvolvido inicialmente para uso em suas próprias aplicações.

O objetivo do Guice é realizar os conceitos do padrão de projeto conhecido como Injeção de Dependências através de anotações e ser bem leve em comparação ao Spring.

Além de não precisar de um grande número de xml (você consegue injetar dependência sem usar nenhum).

GOOGLE GUICE

É ideal principalmente para manter baixo acoplamento entre diferentes camadas do sistema.

Nesse modo o responsável por injetar esse recurso será o container de injeção, fazendo que algumas vezes o programador não saiba de onde vem tal recurso.

Elimina a necessidade do desenvolvedor criar uma instância programaticamente.

GOOGLE GUICE

Vantagens:

- ➔ Sem configuração de XML/.
- ➔ Modularidade : guice-4.2.jar, guice-servlet-4.2.jar, guice-persist-4.2.jar, etc.…
- ➔ Facilidade de uso.
- ➔ Curva de aprendizado.

CONFIGURAÇÃO

Para utilizar o Guice é necessário baixar uma biblioteca e adicioná-la ao projeto.

Pode-se baixá-la como `.jar` ou via Maven pelo repositório do GitHub através do link: http://code.google.com/p/google-guice/.

Para quem usa o Maven, pode baixar adicionando a seguinte configuração no arquivo `pom.xml`.

CONFIGURAÇÃO

Latest release:

```
<dependency>
  <groupId>com.google.inject</groupId>
  <artifactId>guice</artifactId>
  <version>4.2.0</version>
</dependency>
```

MÓDULO

O Módulo é responsável por definir como as dependências são resolvidas, é nele que são criadas as configurações.

```
public class MyAbstractModule extends AbstractModule {

    @Override
    public void configure() {

        bind(LivroDaoInterface.class).to(LivroDao.class);

        bind(LivroServiceInterface.class).to(LivroService.class);
    }

}
```

INJETOR

O Injetor é responsável por descobrir o que deve construir, resolver as dependências e fazer a junção dos mesmos.

```
public class Main {

    public static void main(String[] args) {

        Injector injector = Guice.createInjector(new MyAbstractModule());

        LivroServiceInterface livroServiceInterface
                = injector.getInstance(LivroServiceInterface.class);
    }

}
```

BINDINGS

Bindings: São as configurações realizadas em um módulo.

No Guice existem as seguintes formas:

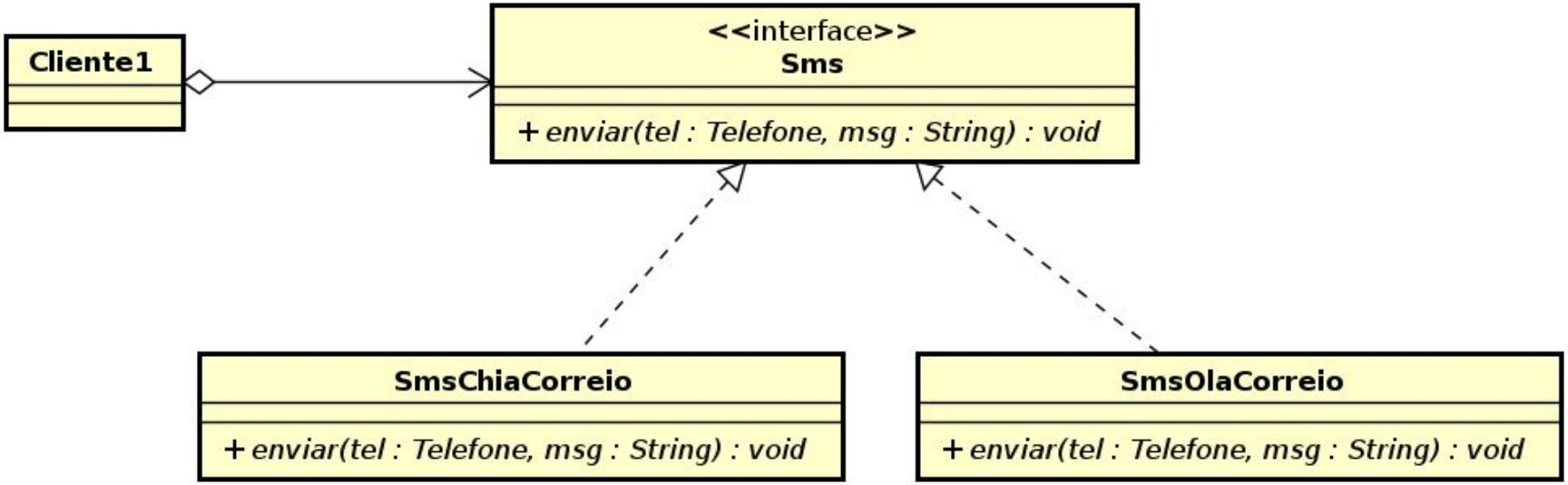
- ➔ Linked Bindings
- ➔ Just-In-Time-Bindings
- ➔ Bindings Annotations
- ➔ Instance Bindings
- ➔ Provider Bindings
- ➔ Constructor Bindings

PROBLEMA 1

Imagine que o sistema **OfertaFácil** precise se comunicar com uma aplicação externa para enviar mensagens SMS para seus clientes.

Atualmente o **OfertaFácil** se comunica com o serviço de SMS da empresa **ChiaCorreio**. Como poderíamos criar uma estrutura de envio de SMS da aplicação **OfertaFácil** onde a troca da empresa que presta esse serviço seja feita de maneira desacoplada, **sem** a necessidade de **instanciarmos(new)** o objeto específico?

DIAGRAMA DA PROBLEMA 1



SOLUÇÃO

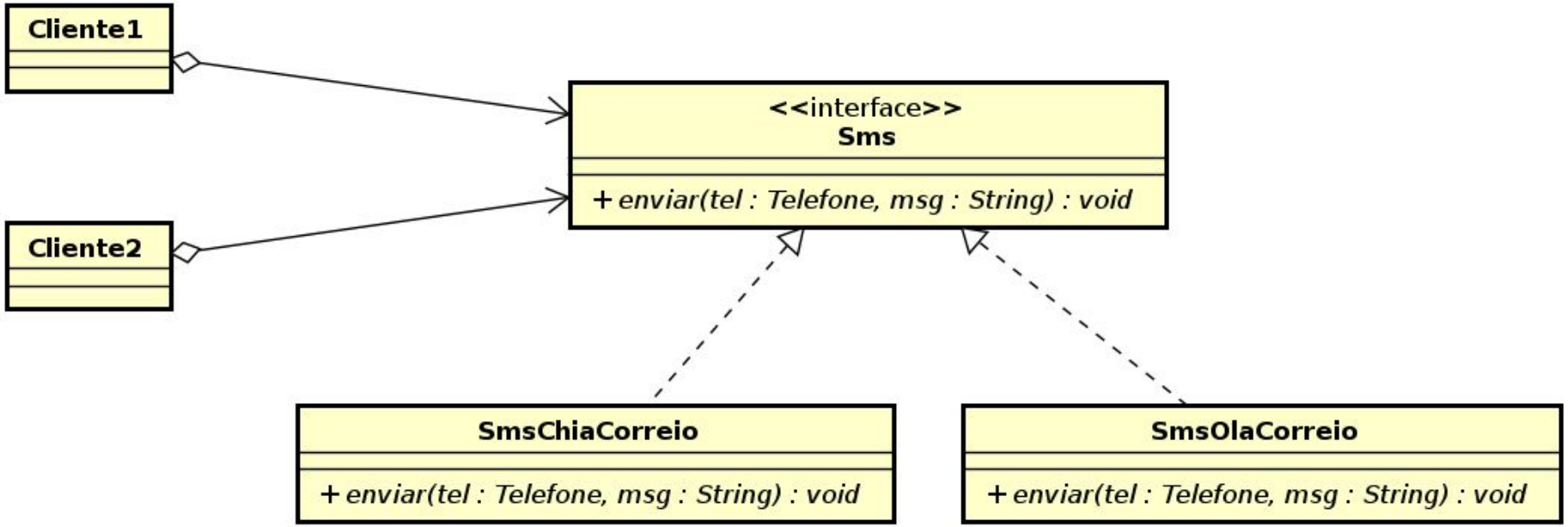
Show me the code, baby!

Link do código:
https://github.com/codeandcry/guice-seminario/tree/master/oferta-facil

PROBLEMA 2

Imagine que no exemplo anterior eu tenha várias classes clientes, onde algumas delas precisam do serviço da empresa ChiaCorreio e outras da empresa OlaCorreio, se usarmos o LinkedBindings não teremos como solucionar esse segundo contexto do problema.

DIAGRAMA DO PROBLEMA 2



SOLUÇÃO

Show me the code, baby!

Link do código:
https://github.com/codeandcry/guice-seminario/tree/master/oferta-facil-qualifiers

QUALIFICADORES

Qualificadores dizem qual classe concreta deve ser injetada quando temos várias implementações de uma mesma interface. Atráves de BindingsAnnotations do Guice podemos criar Qualificadores.

BindingsAnnotations podem ser feitas de duas maneiras:

@AnnotationDefinition

@Named(“String”)

QUALIFICADORES

No exemplo anterior foi usado @Named.

```
public class Cliente1 {
    @Inject
    @Named("ChiaCorreio")
    private Sms sms;
```

```
public class Cliente2 {
    @Inject
    @Named("OlaCorreio")
    private Sms sms;
```

QUALIFICADORES

```
@Override
protected void configure(){
    bind(Sms.class).annotatedWith(Names.named("ChiaCorreio"))
            .to(SmsChiaCorreio.class);
    bind(Sms.class).annotatedWith(Names.named("OlaCorreio"))
            .to(SmsOlaCorreio.class);
}
```

OBJETOS COMPOSTOS

Quando precisamos injetar objetos compostos, que precisam de métodos ou parâmetros para isso. Podemos usar @Provides Methods.

```
@Provides
public Connection provideConnectionPostgres() throws SQLException{
    String url = "jdbc:postgresql://localhost:5432/notas-app";
    String usuario = "postgres";
    String senha = "pgadmin";
    return DriverManager.getConnection(url,usuario,senha);
}
```

LINKED BINDINGS

Uma Linked Binding é a ligação direta entre uma interface e a classe concreta que implementa a mesma.

Ex:

```
bind(InterfaceClasse.class).to(ClasseImpl.class);
```

JUST-IN-TIME-BINDINGS

Para utilizar este tipo de binding é preciso apenas adicionar a anotação @ImplementedBy na interface que será configurada passando como parâmetro a classe concreta.

Ex:

```
@ImplementedBy(Impl.class)
public interface Interface{}
```

BINDINGS ANNOTATIONS

Com esse tipo de Binding é possível fazer com que uma classe que possua uma anotação específica retorne uma instância de uma determinada classe.

Ex:

```
bind(Clas.class).annotatedWith(Anotacao.class).to(Classe.class);
```

INSTANCE BINDINGS

Utilizado quando existe uma instância Singleton no sistema, deste modo, pode-se configurar uma classe para retornar sempre a mesma instância.

Ex:

```
bind(Clas.class).annotatedWith(Integer.class).toInstance(1000);
```

PROVIDER BINDINGS

O Provider Bindings é uma classe responsável por deduzir e instanciar a classe correta.

Ex:

```
private final Provider<Classe> classe;
classe.get();
```

CONSTRUCTOR BINDINGS

Esse tipo de Binding torna possível indicar qual construtor o Guice deve usar ao instanciar a classe.

Ex:

```
bind(I.class).toConstructor(A.class.getConstructor(B.class));
```

REFERÊNCIAS

- https://edermfl.wordpress.com/2016/04/26/injecao-de-dependencia-com-guice/
- https://github.com/google/guice/wiki
- http://www.baeldung.com/guice
- https://www.journaldev.com/2394/java-dependency-injection-design-pattern-example-tutorial
- http://blog.caelum.com.br/ioc-e-di-para-frameworks-mvc/
- http://www.linhadecodigo.com.br/artigo/3418/inversao-de-controle-ioc-e-injecao-de-dependencia-di-diferencas.aspx