フレームワーク基礎実習

03. Spring Boot 基礎

株式会社ジードライブ

今回学ぶこと

- SpringのDI機能
- Lombokの利用

アノテーション

- Springを使用して、システムを開発する際は、アノテーションを多用する
  - アノテーションが付与されているクラスやフィールドは、Springフレームワークによって利用される

アノテーションの例

```
@Bean, @Component, @Autowired
@Configuration, @Controller, @Service
@Repository, @Valid, @RequestMapping …etc
```

  - 例えば、@Configurationというアノテーションが付いているクラスは、設定ファイルとしてSpringに利用される

アノテーションの例

- 以下のアノテーションは依存関係解決の場面で利用される

アノテーション	説明
@Bean	設定ファイル内のメソッドに付与するアノテーション。SpringのDIコンテナによって生成・管理されるオブジェクトの設定を行うことができる
@Component	クラスに付与することで、DIコンテナによるオブジェクトが生成・管理の対象になる。@Beanの簡易版
@Autowired	フィールドやメソッドに付与するアノテーション。DIコンテナによって管理されているオブジェクトの注入を受け付ける
@ComponentScan	@Componentの付与されたクラスを探索する範囲を指定するためのアノテーション

依存関係

- 以下のGreeterクラスはHello型のフィールドをもっており、greetメソッド内で利用している
  ⇒ Helloオブジェクトに**依存**している

Greeterクラス

```
public class Greeter {
  private Hello hello;

  public void greet() {
    hello.sayHi();
  }
}
```

Helloインターフェース

```
public interface Hello {
   void sayHi();
}
```

JapaneseHelloクラス(Helloを実装)

```
public class JapaneseHello implements Hello {
  @Override
  public void sayHi() {
     System.out.println("こんにちは!");
  }
}
```

依存性の注入(DI: Dependency Injection)

- Helloオブジェクトを利用するには、依存関係の解決が必要
- コンストラクタやセッターを通じて、依存しているHelloオブジェクトが代入(注入)される必要がある
  ⇒ 依存性の注入(**DI: Dependency Injection**)と呼ばれる

Dependency Injection(DI)を受け付けるためのコンストラクタ、またはセッターが必要になる

Greeterクラス

```
public class Greeter {
  private Hello hello;

  public Greeter() {}

  public Greeter(Hello hello) {
     this.hello = hello;
  }

  public void setHello(Hello hello) {
     this.hello = hello;
  }

  public void greet() {
    hello.sayHi();
  }
}
```

依存性の注入(DI: Dependency Injection)

- 依存関係を解決しないとNullPointerExceptionが発生する

前頁のGreeterクラスの利用

```
// 依存関係を解決していない
Greeter greeter1 = new Greeter();
greeter1.greet(); // NullPointerExceptionが発生

// コンストラクタを通じて、依存関係を解決(依存オブジェクトを注入)
// コンストラクタ・インジェクションと呼ばれる
Greeter greeter2 = new Greeter(new JapaneseHello());
greeter2.greet(); // 例外は発生しない

// セッターを通じて、依存関係を解決(依存オブジェクトを注入)
// セッター・インジェクションと呼ばれる
Greeter greeter3 = new Greeter();
greeter3.setHello(new JapaneseHello());
greeter3.greet(); // 例外は発生しない
```

Hello型オブジェクトを生成・注入

Hello型オブジェクトを生成・注入

SpringのDI(Dependency Injection)機能

- Springではアノテーションを使い、依存関係を解決することができる
- @Componentアノテーションによって、Springの管理対象になり、生成と注入が自動的に行われるようになる

JapaneseHelloクラス

Springによって、生成・注入が行われるようになるため、new JapaneseHello() の記述が不要になる

```
@Component
public class JapaneseHello implements Hello {
  @Override
  public void sayHi() {
     System.out.println("こんにちは!");
  }
}
```

SpringのDI(Dependency Injection)機能

- DIを受ける側には@Autowiredアノテーションを付与する

Greeterクラス

```
@Component
public class Greeter {

  @Autowired
  private Hello hello;

  public void greet() {
     hello.sayHi();
  }
}
```

Springによって、Hello型のオブジェクトが注入される

SpringのDI(Dependency Injection)機能

- @Autowiredはフィールドではなく、コンストラクタやセッターメソッドに付与してもよい

Greeterクラス

```
@Component
public class Greerter {
  private Hello hello;

  @Autowired
  public void setHello(Hello hello) {
      this.hello = hello;
  }

  public void greet() {
     hello.sayHi();
  }
}
```

セッターに付与するパターン

SpringのDI(Dependency Injection)機能

- コンストラクタに付与する場合、コンストラクタが一つしかなければ、@Autowiredは省略可

Greeterクラス

```
@Component
public class Greerter {
  private final Hello hello;

  @Autowired
  public Greeter(Hello hello) {
      this.hello = hello;
  }

  public void greet() {
     hello.sayHi();
  }
}
```

コンストラクタに付与するパターン
⇒ コンストラクタが一つだけなので省略可

コンストラクタによるDIが推奨されている。
この際、DIを受けるフィールドにfinal
キーワードを付与する

Lombokの利用

Lombokの利用

- Lombokは、アクセッサやコンストラクタなど、何度も書かなければいけない定型のコード(ボイラープレートコード)を自動生成するライブラリ
  - クラスやフィールドに**アノテーション**を付与することで生成するメソッドを制御することができる

Lombokの利用例：引数付きのコンストラクタ、アクセッサが自動生成される

```
@AllArgsConstructor
@Data
public class Item {
     private String name;
     private int price;
}
```

Lombokのアノテーション

- 依存関係にLombokを加えている場合、以下のアノテーションを使用することができる

アノテーション	説明
@NoArgsConstructor	引数なしのコンストラクタを生成する
@AllArgsConstructor	全フィールドを引数にとるコンストラクタを生成する
@RequiredArgsConstructor	finalフィールドに対する引数をとるコンストラクタを作成する
@Builder	ビルダーパターンによるインスタンス生成が可能になる
@Getter	ゲッターメソッドを生成する
@Setter	セッターメソッドを生成する
@ToString	toString()メソッドの生成する
@Data	アクセッサやtoString()などを生成する
@Value	ゲッターメソッドやtoString()などを生成する

Lombok: ビルダーパターン

- ビルダーパターンでは、メソッドチェーンを使い、フィールドに初期値をセットできる

```
@RequiredArgsConstructor
@Builder
@Getter
public class Member {
        private final Integer id;
        private final String name;
        private final Integer age;
}
```

```
// コンストラクタによるインスタンス生成
Member m1 = new Member(1, "山田太郎", 25);

// ビルダーパターンによるインスタンス生成
Member m2 = Member.builder()
        .age(30)
        .name("鈴木次郎")
        .id(2)
        .build();
```

フィールド名が、ビルダーのメソッド名になる
⇒ メソッドの記述順は任意

ビルダーを生成

インスタンスを生成

LombokとDI

- コンストラクタによるDIはLombokのアノテーションによって代用することも可能

Greeterクラス(Lombok未使用)

```
@Component
public class Greeter {
  private final Hello hello;

  @Autowired
  public Greeter(Hello hello) {
      this.hello = hello;
  }

  public void greet() {
     hello.sayHi();
  }
}
```

コンストラクタが一つだけなので省略可

Greeterクラス(Lombok利用)

```
@Component
@RequiredArgsConstructor
public class Greeter {
  private final Hello hello;

  public void greet() {
     hello.sayHi();
  }
}
```

練習問題

- 練習03-1
- 練習03-2