25 DE AUGUST DE 2020 (HTTP://RDRBLOG.COM.BR/JAVA/SPRING/ARQUITETURA-HEXAGONAL-COM-SPRING-BOOT/) RODOLFOCLASS (HTTP://RDRBLOG.COM.BR/AUTHOR/RODOLFOCLASS/)

Arquitetura Hexagonal com Spring Boot

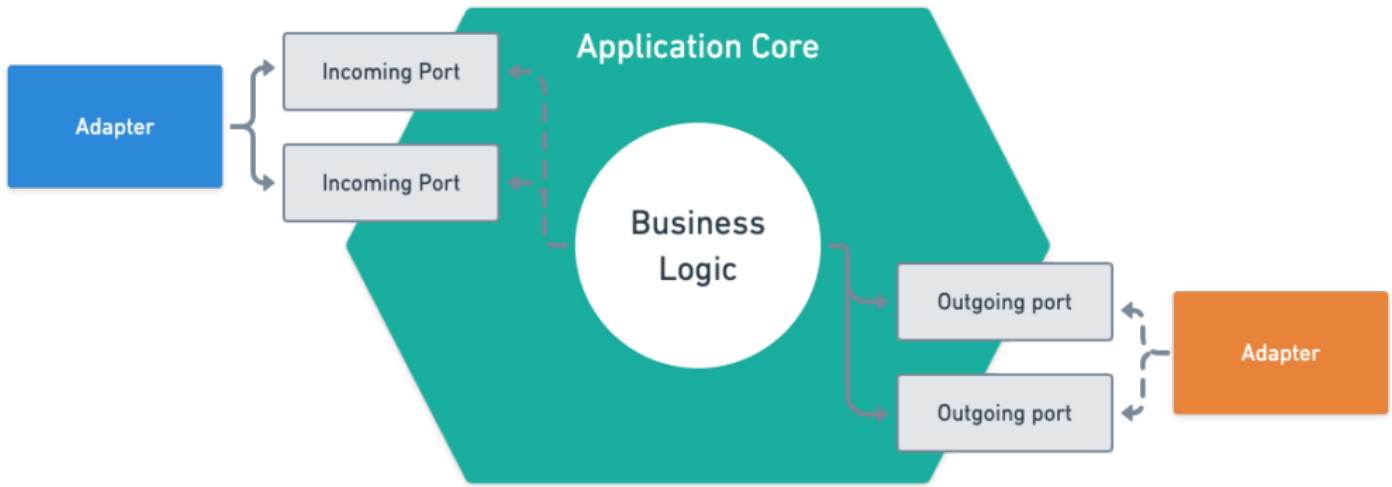
Neste artigo, mostrarei como implementar um aplicativo Spring Boot usando a Arquitetura Hexagonal.

Vamos construir uma simulação de conta bancária com operações de *depósito* e *saque* expostas por meio de endpoints REST.

Arquitetura Hexagonal

A arquitetura hexagonal é um estilo arquitetônico que **se concentra em manter a lógica de negócios separada das preocupações externas** .

O núcleo de negócios interage com outros componentes por meio de portas e adaptadores. Dessa forma, podemos mudar as tecnologias subjacentes sem ter que modificar o núcleo do aplicativo.



-a-objetos/)

RECENT POSTS

Java 15!!! O que vem ai? (http://rdrblog.com.br/java/java-15-o-que-vem-ai/)

Arquitetura Hexagonal com Spring Boot (http://rdrblog.com.br/java/spring/arquitetura-hexagonal-com-spring-boot/)

O que é Spring Boot (http://rdrblog.com.br/java/o-que-e-spring-boot/)

Trazendo Java para o futuro nativo do Kubernetes com o Quarkus (http://rdrblog.com.br/java/trazendo-java-para-o-futuro-nativo-do-kubernetes-com-o-quarkus/)

Tratamento de exceções em Java Streams (http://rdrblog.com.br/java/tratamento-de-excecoes-em-java-streams/)

Application Core

Modelo de Domínio

Vamos começar com o modelo de domínio. Sua principal responsabilidade é modelar as regras de negócios. Ele também verifica se os objetos estão sempre em um estado válido:

```
Java
public class BankAccount {

    private Long id;
    private BigDecimal balance;

    // Constructor

    public boolean withdraw(BigDecimal amount) {
        if(balance.compareTo(amount) < 0) {
            return false;
        }

        balance = balance.subtract(amount);
        return true;
    }

    public void deposit(BigDecimal amount) {
        balance = balance.add(amount);
    }
```

```
}
```

O modelo de domínio não deve depender de nenhuma tecnologia específica. Essa é a razão pela qual você não encontrará anotações do Spring aqui.

Portos

Agora é a hora de fazer com que nossa lógica de negócios interaja com o mundo externo. Para conseguir isso, vamos apresentar algumas portas.

Primeiro, vamos definir 2 portas de entrada. **Eles são usados por componentes externos para chamar nosso aplicativo** . Nesse caso, teremos um por caso de uso. Um para *depósito* :

Java

```
public interface DepositUseCase {
      void deposit(Long id, BigDecimal amount);
}
```

E um para *retirada* :

Java

```
public interface WithdrawUseCase {
      boolean withdraw(Long id, BigDecimal amount);
}
```

Serviço

A seguir, criaremos um serviço para unir todas as peças e conduzir a execução:

```java
public class BankAccountService implements DepositUseCase, WithdrawUseCase {

    private LoadAccountPort loadAccountPort;
    private SaveAccountPort saveAccountPort;

    // Constructor

    @Override
    public void deposit(Long id, BigDecimal amount) {
        BankAccount account = loadAccountPort.load(id)
                .orElseThrow(NoSuchElementException::new);

        account.deposit(amount);

        saveAccountPort.save(account);
    }

    @Override
    public boolean withdraw(Long id, BigDecimal amount) {
        BankAccount account = loadAccountPort.load(id)
                .orElseThrow(NoSuchElementException::new);
```

```
            boolean hasWithdrawn = account.withdraw(amount);

            if(hasWithdrawn) {
                saveAccountPort.save(account);
            }
            return hasWithdrawn;
        }
}
```

Observe como o serviço implementa as portas de entrada. Em cada método, ele usa a porta de *carregamento* para buscar a conta no banco de dados. Em seguida, realiza as alterações no modelo de domínio. E, finalmente, ele salva essas alterações por meio da porta *Salvar* .

Adaptadores

Rede

Para completar nosso aplicativo, precisamos fornecer implementações para as portas definidas. Chamamos esses adaptadores.

Para as interações de entrada, criaremos um controlador REST:

```
   Java
@RestController
@RequestMapping("/account")
public class BankAccountController {

      private final DepositUseCase depositUseCase;
```

```
    private final WithdrawUseCase withdrawUseCase;

    // Constructor

    @PostMapping(value = "/{id}/deposit/{amount}")
    void deposit(@PathVariable final Long id, @PathVariable final BigDecimal amo
        depositUseCase.deposit(id, amount);
    }

    @PostMapping(value = "/{id}/withdraw/{amount}")
    void withdraw(@PathVariable final Long id, @PathVariable final BigDecimal ar
        withdrawUseCase.withdraw(id, amount);
    }
}
```

O controlador usa as portas definidas para fazer chamadas ao núcleo do aplicativo.

Persistência

Para a camada de persistência, usaremos Mongo DB por meio do Spring Data:

```
Java
public interface SpringDataBankAccountRepository extends MongoRepository<BankAccou
}
```

Além disso, criaremos uma classe *BankAccountRepository* que conecta as portas de saída com *SpringDataBankAccountRepository* :

Java

```
@Component
public class BankAccountRepository implements LoadAccountPort, SaveAccountPort {

    private SpringDataBankAccountRepository repository;

    // Constructor

    @Override
    public Optional<BankAccount> load(Long id) {
        return repository.findById(id);
    }

    @Override
    public void save(BankAccount bankAccount) {
        repository.save(bankAccount);
    }
}
```

A infraestrutura

Finalmente, precisamos dizer ao Spring para expor o *BankAccountService* como um bean, para que ele possa ser injetado no controlador:

Java

```
@Configuration
@ComponentScan(basePackageClasses = HexagonalApplication.class)
```

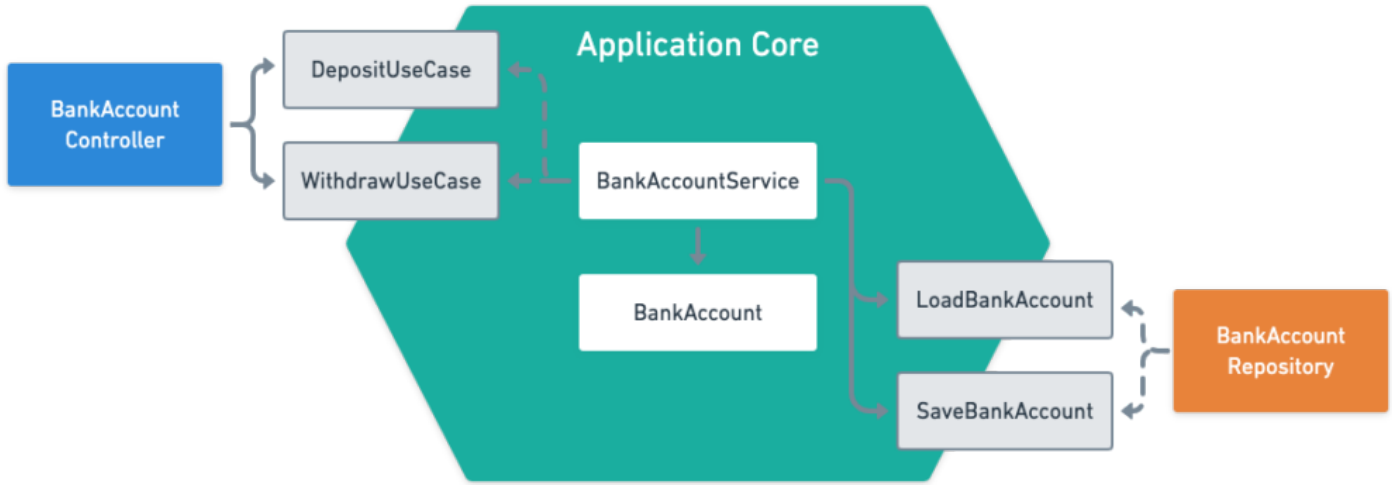
```
public class BeanConfiguration {

      @Bean
      BankAccountService bankAccountService(BankAccountRepository repository) {
          return new BankAccountService(repository, repository);
      }
}
```

Definir os beans na camada de Adaptadores nos ajuda a manter o código de infraestrutura desacoplado da lógica de negócios.

Conclusão

Neste artigo, vimos como implementar um aplicativo usando Arquitetura Hexagonal e Spring Boot. É assim que o sistema acaba ficando:



O exemplo desse artigo esta no meu GitHub (https://github.com/roddramos/hexagonal-architecture-spring).

ARQUITETURA DE SOFTWARE (HTTP://RDRBLOG.COM.BR/CATEGORY/ARQUITETURA-DE-SOFTWARE/) , JAVA (HTTP://RDRBLOG.COM.BR/CATEGORY/JAVA/) , SPRING (HTTP://RDRBLOG.COM.BR/CATEGORY/JAVA/SPRING/)

O que é Spring Boot (http://rdrblog.com.br/java/o-que-e-spring-boot/)

Java 15!!! O que vem ai? (http://rdrblog.com.br/java/java-15-o-que-vem-ai/)

Leave a Reply

*Your email address will not be published. Required fields are marked **

Comment *

Name *

Email *

Website

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

This site uses Akismet to reduce spam. Learn how your comment data is processed (https://akismet.com/privacy/).