

true

Alexandre Fabian

Chain of Responsibility

Chain of Responsibility

É um padrão de projeto comportamental

- 1 Permite que se passe requisições por uma corrente de processamento.
- 2 O Padrão é composto de um objeto de cliente e um grupo de handlers.

- 3 Evita o acoplamento do remetente ao receptor, ao dar a mais de um objeto a oportunidade de tratar a solução (GAMMA et al., 2000)

- Cada Handler trata (ou não) a requisição.
- Decide se passa para o próximo ou quebra a corrente.

- Reduz a complexidade do problema.
- Deixa os objetos mais simples.

Quando Aplicar

- Quando mais de um objeto precisa processar uma requisição;
- os handlers precisam de uma ordem especifica para serem executados;
- os handlers utilizados e sua ordem podem ser mudados dinamicamente.

- É possível controlar a ordem que a requisição irá passar pelos handlers.
- Princípio de Responsabilidade Única.
- Princípio do Aberto/Fechado.

- Algumas requisições podem ficar sem tratamento.

Como aplicar

- Declarar uma **interface** para o Handler;
- um Handler **base** com código comum aos demais;
- os Handlers **concretos**;
- e o **Cliente** que irá compor a corrente.

Sobre o exemplo:

- Revision Web Service: Serviço para revisão de exercício de programação realizado no Github e registro de nota no Moodle.

Exemplo

```
/**
 * Creates a Chain of checkers
 *
 * @return The first Checker
 */
private Checker createGithubChain(){
    moodleUser.setNextChecker(enrolled);
    enrolled.setNextChecker(repository);
    repository.setNextChecker(fork);
    fork.setNextChecker(testChange);
    testChange.setNextChecker(moodleSend);
    return moodleUser;
}
```

Figure 1: Chain do Projeto Revision

Exemplo

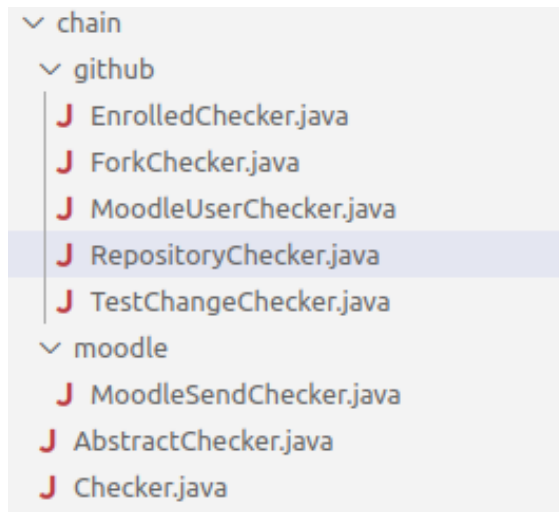


Figure 2: Lista de Checkers

Exemplo

```
import org.springframework.stereotype.Component;

@Component
@ApplicationScoped
public class RepositoryChecker extends AbstractChecker implements Checker {

    private static final Logger LOGGER = Logger.getLogger(RepositoryChecker.class.getName());

    @Override
    public boolean check(Map<String, String> input) {
        LOGGER.info("RepositoryChecker");

        boolean result = false;

        String moodleAssign = input.get("moodleAssign");
        String githubLogin = getGithubLogin(input.get("githubProfileURL"));

        // Module in this case will be the assign
        // This step discovers the course id and the instance id (the data base id)
        // The instance id is necessary to update the grade
        Module module = getCourseModule(moodleAssign);

        // Returns the courses and the assigns
        // We need this step to retrieve the assign intro (description)
        List<Course> courses = getMoodleCourse(module);

        // Gets the assign intro (description)
        String intro = this.getAssignIntro(courses, moodleAssign);

        // Get the YAML from the assign
        if (intro != null) {
            Map<String, String> config = this.getAssignConfig(intro);
            if (config != null) {
                List<WorkflowAction> actions = github.getRuns(githubLogin, config.get("repo"), config.get("workflow"));
                Run run = actions.getLatestRun();
                // Verifies if the latest run was a success and the repository is a fork
                if (run.getConclusion().equalsIgnoreCase("success") && run.getRepository().isFork()) {
                    LOGGER.info("Os testes passaram e é um fork");
                    result = this.getNextChecker().check(input);
                }
            }
        }
        else {
            String message = messages.getString("RepositoryChecker.repo");
            LOGGER.log(Level.WARNING, message);
            throw new RevisionServiceException(message, Response.Status.BAD_REQUEST);
        }
    }

    return result;
}
```

Figure 3: Exemplo de Checker Concreto

Referências

- <https://github.com/orion-services/revision>
- SHVETS, Alexander. Dive Into Design Patterns. Kamianets-Podilskyi, Ucrânia. Refactoring.Guru, 2019. 409 p.
- GAMMA, Erich; JOHNSON, Ralph; HELM, Richard; VLISSIDES, John. Padrões de Projetos:oluções reutilizáveis de software orientados a objetos. São Paulo: Bookman, 2000. 360 p. Tradução Luiz A. Meirelles Salgado.