45426- Teste e Qualidade de Software

E1

Exame de Época Normal – 2021-06-29 10h00. Duração: 50min (+10 tolerância).

NOME: Nr.MEC:

Responda na grelha, assinalando a opção (mais) verdadeira, exceto se a perguntar fornecer outra orientação. As não-respostas valem zero pontos. **Respostas erradas descontam** 1/5 da cotação da pergunta; as respostas assinaladas de forma ambígua serão consideradas não-respostas. Cotação da escolha múltipla: 15v.

Grelha de resposta → Perguntas 1 a 20: escrever a opção de resposta em maiúscula, A, B, C, D, E, ou deixar vazio.

P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13
P14	**P15**	**P16**	**P17**	**P18**	**P19**	**P20**	**P21**	**P22**	**P23**	**P24**	**P25**	

P1

"Uma das maiores vantagens de usar o TDD é que estamos a fazer pequenos incrementos ao escrever código e, portanto, os problemas são mais fáceis de corrigir, uma vez que apenas abordam um pedaço de código limitado." [In: Relatório de projeto de TQS]

A vantagem referida pode ser considerada real e fundamentada?

Selecione uma opção:

(A) Sim, o código de produção é feito na medida do suficiente para satisfazer os testes e a causa dos eventuais problemas é fácil de localizar.
(B) Não, se cada teste só incidir sobre pedaços limitados do código a correção de problemas é ilusória, porque não verifica a integração dos módulos.
(C) Sim, o TDD previne a inserção de falhas no código.
(D) Sim, em TDD, é frequente reutilizar pedaços limitados do código dos próprios testes para escrever o código de produção.
(E) Não, porque é preciso escrever bastante mais código em produção do que aquele que é necessário para passar os testes.

P2

A abordagem BDD preconiza a realização de testes numa estratégia de QA, relacionados diretamente com as *user stories*. Qual das seguintes caraterísticas NÃO se aplica ao BDD?

Selecione uma opção:

(A) Os (novos) cenários tornam-se mais fáceis de implementar à medida que mais "passos" ficam implementados, pois há a possibilidade de partilhar/reusar passos comuns.
(B) Facilita a compreensão do comportamento esperado do software, ao fomentar a colaboração entre a equipa técnica e a equipa do "negócio".
(C) É suportada por uma linguagem natural que permite descrever o teste na área do domínio do problema, entendida pela equipa alargada.
(D) Dispensa a utilização de outras técnicas de especificação de requisitos, uma vez que permite centralizar a especificação do produto na descrição da "*feature*", que é sujeita a controlo de versões, com o restante código.
(E) As "*features*" podem ser chamadas de "especificações executáveis" porque alimentam a realização de testes (em código).

P3

A utilização de *personas* (personagens) ajuda a focar a criação de valor nas motivações de utilizadores concretos e cria contextos reutilizáveis para escrever as histórias. Como é que os processos de QA podem beneficiar do recurso às *personas*?

Selecione uma opção:

(A) Embora as personas ajudem a concretizar quem beneficia do produto, é preferível usar os papéis do utilizador diante do sistema (e.g.: um cliente, um aluno) para não enviesar os critérios de aceitação.
(B) As personas fixam um conjunto de caraterísticas do utilizador e ajudam a criar exemplos de utilização coerentes, na escrita dos critérios de aceitação.
(C) Apesar de serem úteis no processo de descoberta de requisitos e fixação das histórias, o conceito de persona não tem continuidade para a definição de testes funcionais.
(D) As personas podem ser usadas parar criar os ramos ("branches") de um repositório (um "branch" para cada persona), facilitando a distribuição do trabalho e a revisão pelos pares.
(E) As personas são usadas para os projetos que seguem BDD, para incluir nos ficheiros de "features", com os exemplos de utilização.

P4

É uma boa prática de QA incluir na lista de condições para concluir um incremento ("*definition of done*") a obrigação do programador suplementar o seu código de produção com testes?

Selecione uma opção:

(A) Sim, porque os ambientes de CI não podem realizar as respetivas *pipelines* se não houver testes no projeto.
(B) Não; se a equipa adotar a revisão de código pelos pares (*peer code review*), os testes são facultativos e apenas necessários em alguns casos mais complexos.
(C) Sim, porque a possibilidade de fazer entrega contínua depende de um elevado grau de confiança na qualidade do código, com níveis de cobertura muito próximos de 100%.
(D) Sim, porque o programador tem conhecimento da lógica interna do módulo e assim forma-se cumulativamente uma "rede de segurança" para prevenir futuras regressões.
(E) Não; existem várias ferramentas de análise de qualidade do código que compensam a eventual falta de testes e há ocasiões em que o programador tem de entregar código e não pode perder tempo com os testes.

P5

O Gitflow é um fluxo de trabalho baseado em Git que foi primeiramente defendido e popularizado por Vincent Driessen. Quais as regras distintivas desta abordagem?

Selecione uma opção:

(A) Usa o ramo "master" para integrar (*merge*) os novos incrementos; as novas funcionalidades devem ser desenvolvidas em ramos próprios, designados segundo o padrão "feature/my-feature"

(B) É uma variação de um modelo de "feature branch", mas em prevê que a história global do projeto seja desenvolvida em dois ramos: o "master" (usado para marcar versões principais do projeto) e "develop" (para a integração regular dos incrementos).

(C) Não é recomendado para todos os projetos, uma vez que não contempla, no processo de trabalho da equipa, a inclusão de "merge/pull requests" para a linha do "master" de forma natural.

(D) Cada programador deve criar um *branch* próprio, fazer alterações no seu *branch* "privado", e integrar os incrementos no *master* partilhado.

(E) Não é recomendado para todos os projetos, uma vez que tem extensões próprias que podem ser incompatíveis com os repositórios Git comuns (e.g.: "git flow release start 0.1") ou em que a linha principal não se chama *master* (e.g.: "master" vs "main").

P6

Considerando o papel dos testes de integração, identifique a afirmação FALSA:

Selecione uma opção:

(A) Os testes de integração podem ser otimizados recorrendo ao uso generalizado de objetos de substituição (mocks) em lugar das dependências.

(B) Os testes de integração, para além da unidade sob teste, usam componentes ou infraestruturas adicionais como a rede, base de dados ou sistema de ficheiros.

(C) Os testes de integração podem ter de esperar que certas etapas do ciclo de CI estejam concluídas, para garantir que houve etapas prévias que prepararam o ambiente necessário para os testes de integração.

(D) Os testes de integração devem ser utilizados para testar as arquiteturas de microsserviços

(E) Quando um teste de integração falha, isso indicia um problema na articulação dos componentes ou na disponibilidade dos serviços.

P7

Nem sempre é necessário, ou conveniente, testar uma unidade de software (ou partes). Quando é que pode ser contraproducente/desaconselhável escrever testes (unitários)?

Selecione uma opção:

(A) No código que ainda não foi revisto; é útil aguardar pelo feedback da revisão do código pelos pares para então escrever os testes.

(B) O código que foi automaticamente gerado (e.g.: vários "toString" e "equals"), também deve ser testados para atingir a cobertura próxima de 100%.

(C) Se a equipa não inclui a avaliação de cobertura nos patamares de qualidade requeridos (*quality gate*), é contraproducente estar a dedicar tempo à escrita de testes unitários.

(D) É sempre necessário e útil escrever testes unitários para a totalidade do código de um projeto.

(E) Objetos que não incluem lógica relevante, como entidades (de persistência) geradas, dispensam a realização de testes.

P8

"Os testes de sistema irão ser conduzidos pelos *stakeholders*, com base na informação presente no manual de qualidade do software e no relatório de especificações do produto. Estes testes irão indicar se cada componente é capaz de satisfazer os requisitos definidos nas especificações do produto." [In: relatório de projeto de TQS]

Esta declaração apresenta vários problemas, assinalados a seguir. Qual das afirmações é FALSA?

Selecione uma opção:

(A) Os testes são automatizados e, por isso, os *stakeholders* nunca participam na realização de testes.

(B) Os testes de sistema são, principalmente, orientados para a qualidade interna do produto e não para as necessidades dos utilizadores.

(C) Os testes de sistema servem para validar o sistema globalmente, não cada componente por si.

(D) O Manual de Qualidade não tem informação suficiente para determinar os testes de sistema que podem ser usados na verificação de um produto.

(E) Os *stakeholders* não realizam testes de sistema, apesar de participar na elaboração de critérios de aceitação.

P9

Em que situação é mais provável que o programador recorra ao uso de objetos de substituição (*mocks*) num plano de teste?

Selecione uma opção:

(A) A equipa adotou uma metodologia BDD para os testes.

(B) A unidade sob teste ainda não está pronta (implementada) e é necessário criar uma implementação provisória do seu comportamento.

(C) A unidade sob teste é um serviço (*endpoint*) de uma API REST.

(D) A unidade sob teste utiliza uma base de dados em memória.

(E) A unidade sob teste apresenta uma dependência de um serviço remoto, com um resultado variável.

P10

Considere a seguinte descrição sobre um teste incluído numa aplicação Spring Boot: "este teste procura confirmar que os objetos JSON da resposta estão a ser mapeados corretamente quando é usado um objeto válido no pedido, e se não forem válidos os pedidos, que estamos a enviar o erro HTTP correto, juntamente com uma razão descritiva para a falha." [In: relatório de um projeto de TQS]

Qual o tipo de teste (predefinido na Spring Boot) mais eficiente/adequado para o propósito enunciado?

Selecione uma opção:

(A) @MockMvc

(B) @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)

(C) @SpringBootTest

(D) @ExtendWith(MockitoExtension.class)

(E) @WebMvcTest

P11

M. Fowler apresenta dez práticas recomendadas para a preparação de um sistema de Integração Contínua. Nesse contexto, qual das seguintes é ERRADA:

Selecione uma opção:

(A) Todos os membros da equipa têm acesso imediato ao feedback do estado das *builds*.
(B) Todos os membros da equipa devem fazer a entrega frequente de incrementos para a linha principal do desenvolvimento, por exemplo, diariamente.
(C) Devem existir testes feitos em ambientes que mimetizam as condições de produção.
(D) As builds devem ser feitas no ambiente de integração com uma periodicidade diária e os resultados distribuídos por um sistema de feedback rápido (e.g.: Slack, Discord).
(E) Devem existir formas automáticas de fazer as instalações, para configurar e montar diferentes ambientes de teste.

P12

É muito frequente a utilização de virtualização por *containers* (contentores), como o Docker, em ambientes de CI/CD.

Relativamente ao contributo dos *containers* nos processos de QA, qual das seguintes afirmações é ERRADA:

Selecione uma opção:

(A) A definição dos *containers* pode ser sujeita a controlo de versões juntamente com o resto do código e recursos do projeto, de forma que a configuração da infraestrutura evolua com história do próprio projeto.
(B) Os containers podem ser geridos num repositório (um *registry*), de modo que *containers* criados em certas etapas do processo de CI/CD podem ser reusados em outras subsequentes.
(C) A generalidade dos testes pode beneficiar do uso de containers como ambiente de execução, designadamente com a utilização da biblioteca "Test Containers.
(D) As ferramentas de CI/CD suportam operações com *containers*, na definição dos respetivos *pipelines* de CI/CD
(E) Os *containers* oferecem uma forma natural e eficiente de isolar elementos da arquitetura, que podem ser ativados ou desativados pelos processos de CI/CD

P13

A expressão "*pipeline as code*", associada, por exemplo, aos processos de CI do GitLab, significa que:

Selecione uma opção:

(A) O *pipeline* é compilado, como o resto do código fonte, pela ferramenta de montagem/*build* usada no projeto (e.g.: Maven).
(B) É possível executar condicionalmente algumas etapas do *pipeline*, fazendo-as depender do sucesso das antecedentes.
(C) Existe uma definição do pipeline escrita numa sintaxe própria e essa configuração está sujeita ao controlo de versões, no repositório onde está o restante código do projeto.
(D) Um *pipeline* de CI inclui a compilação de código fonte e produz artefactos (binários) que são partilhados na equipa.
(E) Os passos do pipeline são executados quando é entregue novo código fonte no repositório associado (evento de "push").

P14

A complexidade ciclomática do código influencia a facilidade com que pode ser lido e mantido. Que fatores podem alterar a métrica de complexidade, tal como é definida no SonarQube?

Selecione uma opção:

(A) Número de variáveis declaradas numa classe.
(B) Número de métodos públicos de uma classe.
(C) Número de iterações realizadas em ciclos (repetições de um ciclo).
(D) Ocorrência de instruções que dividem o controlo de fluxo, tais como if, for, while, case.
(E) Tamanho de uma classe (número de linhas).

P15

Um relatório de cobertura, por exemplo, gerado pelo Jacoco, apresenta diversas dimensões segundo as quais a cobertura pode ser calculada (e.g.: métodos, instruções, linhas, ramos alternativos). Qual das seguintes opções é uma descrição adequada do conceito de cobertura de instruções do código?

Selecione uma opção:

(A) É uma métrica que determina a relação das instruções que foram executadas num conjunto de testes, em relação ao total de instruções do código analisado.
(B) É uma métrica que traduz a percentagem de código que foi inspecionado pelo ambiente de análise estática (e.g.: SonarQube) em relação à totalidade do projeto.
(C) É uma métrica que traduz o número de métodos (funções) que foram invocados nos testes, em relação à totalidade dos métodos existentes no código.
(D) É uma métrica utilizada para medir a percentagem de linhas de código fonte que são realmente executadas.
(E) É uma métrica usada para medir a percentagem de testes que foram executados com sucesso.

P16

O guia de estilo para os colaborados do Android Open Source Project indica que a situação ilustrada no exemplo junto não é aceitável. Em termos gerais, qual é a alternativa preferencial para resolver esta situação?

```
void setServerPort(String value){
    try{
        serverPort =Integer.parseInt(value);
    } catch( NumberFormatException e){ }
}
```

Selecione uma opção:

(A) Registar a exceção no *log* da aplicação.
(B) Receber a exceção e lançar uma nova ocorrência da exceção genérica (Exception).
(C) Receber a exceção e lançar uma nova ocorrência de RuntimeException.
(D) Ignorar a exceção, sem a tratar, desde que se inclua um comentário no código a documentar a razão para o fazer.
(E) Lançar uma nova exceção, mas apropriada ao nível de abstração da operação (com a semântica adequada).

P17

O suporte para testes existente no Spring Boot integra bem com o *framework* Mockito. O que seria uma situação natural para usar o Mockito no teste de componentes da Spring Boot?

Selecione uma opção:

(A) O Mockito pode ser usado num teste @DataJpaTest de um componente @Repository para definir os resultados esperados das *queries* que seriam colocadas à base de dados.
(B) O Mockito pode ser usado num teste @DataJpaTest de um componente @Service para substituir as chamadas à base de dados por respostas sintetizadas, fazendo mock do componente repositório.
(C) O Mockito é um framework para usar no contexto de testes unitários e, genericamente, não é útil em testes

da SpringBoot que precisam de injetar componentes adicionais ("beans").

(D) O Mockito pode ser usado num teste @WebMvcTest restrito a um componente do tipo @RestController, para substituir a lógica do serviço, injetando um "bean" sintetizado (via @Spy).

(E) O Mockito pode ser usado num teste unitário de um módulo, para predefinir os resultados do RestTemplate, quando o módulo depende da invocação de métodos de uma API externa.

P18

O excerto ilustrado consta de um ficheiro, de um projeto de alunos de TQS. O que é que este excerto mostra?

```
Scenario Outline: Logged user wants to search
products category
   Given at least one <category> product exists
   And <user> is in the home page
   When <user> clicks the search button
   Then he sees the search results page
   And results are for <category>
   And number of results is more than zero

   Examples:
     | user      | category     |
     | Francisco | BOOKS        |
```

Selecione uma opção:

(A) A definição parcial de uma "feature" compatível com o *framework* Cucumber, com três pós-condições avaliadas em "steps" próprios.

(B) O resultado de um teste com Cucumber, em que o utilizador "Francisco" pesquisou as entradas da categoria "BOOKS"

(C) Uma "feature" que leva à ativação, entre outros, do método de teste com a assinatura "`@Then("number of results is more than zero") public void number_of_results_is_more_than_zero(int expectedValue) {}`"

(D) O log produzido pelo Hibernate quando, na aplicação, foi solicitado a um componente do tipo "repositório" a consulta de dados da entidade utilizador, com uma projeção nos campos "user" e "category".

(E) A definição de um teste funcional do Selenium, com recurso à sintaxe do Gerkin.

P19

Podemos beneficiar da utilização do *framework* Mockito para escrever testes da camada de serviços (i.e., componentes do tipo @Service), tendo presente a arquitetura comum de uma aplicação Spring Boot.

Qual das situações corresponde a uma (boa) prática típica?

Selecione uma opção:

(A) Utilizar @InjectMock no serviço sob teste e @Mock dos componentes (de tipo) repositório requeridos.

(B) Fazer @Mock do serviço que se pretende testar e utilizar o TestEntityManager para realizar as operações sobre a base de dados.

(C) Marcar a classe de teste com @Service, o que satisfaz automaticamente as dependências assinaladas com @InjectMock (i.e., obtém "mocks" dos repositórios necessários).

(D) Anotar o serviço que se pretende testar com @WebMvcTest e as dependências necessárias com @MockMvc (especialmente os repositórios).

(E) Utilizar TestRestTemplate para aceder à interface (programática) do serviço que se pretende testar e fazer @Mock dos componentes de tipo repositório requeridos.

P20

O excerto de código na Figura 1 mostra um cenário de teste usando práticas comuns da Spring Boot.

Qual é a situação que se está a testar com este código (isto é, o que se pode concluir se o teste passar)?

Selecione uma opção:

(A) O GreetingService contém um método de "greet()" que devolve a concatenação de "Hello" e o argumento passado na invovação do endpoint "/greeting".

(B) O GreetingService#greet() retorna uma resposta JSON que inclui sempre "Hello, Mock".

(C) O GreetingService usa uma instância mock do GreetingController que responde a pedidos HTTP no endpoint "/greeting".

(D) Existe um endpoint "/greeting", mapeado por algum método da classe GreetingController, que, por sua vez, usa o método GreetingService#greet()

(E) O GreetingController define um mapeamento para o caminho "/greeting" que deve ser invocado com um argumento do tipo String, contendo "Hello, Mock".

P21

Os testes unitários devem ser atómicos, sucintos e independentes.

Neste contexto, identifique a declaração FALSA.

Selecione uma opção:

(A) O resultado de um teste unitário não deve depender dos resultados de outro teste prévio.

(B) Para evitar a proliferação de métodos numa classe de testes, o programador deve avaliar a oportunidade de verificar várias condições em cada método.

(C) Um teste unitário centra-se no teste de uma única função, o que torna clara a origem do problema, caso o teste falhe.

(D) Manter os testes unitários pequenos ajuda a que os ciclos de CI sejam eficientes.

(E) Os testes unitários devem ser independentes da ordem pela qual são executados, e até independentes do facto de os testes anteriores terem passado ou não.

P22

Um teste típico escrito com Selenium WebDriver é um teste que:

Selecione uma opção:

(A) Exercita parte de um sistema integrado, realizando o teste sobre a interface do utilizador, na web.

(B) Invoca ações numa aplicação web, controlando o browser Firefox ou Chrome (i.e., o WebDriver abstrato é estendido ou por FirefoxDriver ou por ChromeDriver).

(C) É usado para realizar os testes de aceitação no servidor de integração contínua, em lugar dos utilizadores concretos.

(D) Permite fazer mock das dependência dos serviços (componentes @Service) para testar a camada web de modo isolado.

(E) Simula a carga criada por um conjunto configurável de sessões de utilizadores, a aceder à camada da interface web.

P23

Identifique, nos casos listados, uma situação em que recomendaria o uso do *framework* "Test Containers" na implementação de testes.

(A) O teste unitário deve ser executado num ambiente específico, diferente do ambiente em que o programador está a trabalhar.
(B) O teste de integração deve ligar-se a uma base de dados, configurada num ambiente de pré-produção.
(C) O teste de automação na web vai correr num servidor, sem ambiente gráfico.
(D) Há um subconjunto de testes de integração que devem testar a camada de acesso a dados sobre vários motores de bases de dados, de forma tão realista quanto possível e.g.: dialetos de SQL,...).
(E) A aplicação em desenvolvimento utiliza *containers* Docker para a instalação do ambiente de produção, designadamente com "Docker compose".

P24

Como é que o CI/CD pode ajudar nas alturas em que há "picos" de trabalho/pressão pelos resultados, por exemplo, nas alturas próximas das entregas dos *milestones*?

(A) Nesses casos, o *pipeline* de CI/CD pode revelar-se um "bottleneck", e pode ser conveniente aligeirar os patamares de qualidade requeridos (*quality gates).*
(B) O CI/CD com a aplicação de testes e patamares de qualidade (quality gates) de forma automática e obrigatória, é determinante para prevenir a degradação da qualidade dos produtos.
(C) O CI deve ser simplificado, limitando-se os critérios da análise estática, mas não dos testes, de modo a não atrasar a entrega de incrementos.
(D) Quando a equipa está familiarizada com as práticas de CI/CD não ocorrem picos de entrega; o conceito de "contínuo" significa que não flutuações nas entregas.
(E) Nas alturas de pico de trabalho, é quando é mais necessário poder automatizar os testes de desempenho, no pipeline de CI/CD.

P25

Considere que está a rever o código de outro programador que deve implementar um módulo de uma *cache* genérica (Figura 3), que observa um certo tempo de vida das entradas (TTL).

Aconselharia algum dos seguintes "*refactorings*"?

(A) O método *get(key)* não deve provocar uma *NullPointerException*, porque o módulo que o invocou não tem como saber se a chave procurada existe ou não na estrutura.
(B) Os métodos apresentados não devem gerar NullPointerException; em alternativa, devem registar esses eventos num *logger* da aplicação.
(C) O método*putOrTouch* deve ser desagregado, de modo a que os respetivos testes unitários possam ser independentes entre si.
(D) O tempo fixo de expiração definido na interface (ENTRY_TTL) é um problema, uma vez que isso tira flexibilidade, designadamente para a escrita de testes que verificam se uma entrada já expirou.
(E) O código está bem; não oferece oportunidades relevantes de *refactoring*.

```
@WebMvcTest(GreetingController.class)
public class WebMockTest {

        @Autowired
        private MockMvc mockMvc;

        @MockBean
        private GreetingService service;

        @Test
        public void greetingShouldReturnMessageFromService() throws Exception {
                when(service.greet()).thenReturn("Hello, Mock");

this.mockMvc.perform(get("/greeting")).andDo(print()).andExpect(status().isOk())

.andExpect(content().string(containsString("Hello, Mock")));
        }
}
```

Figura 1- Teste Spring Boot.

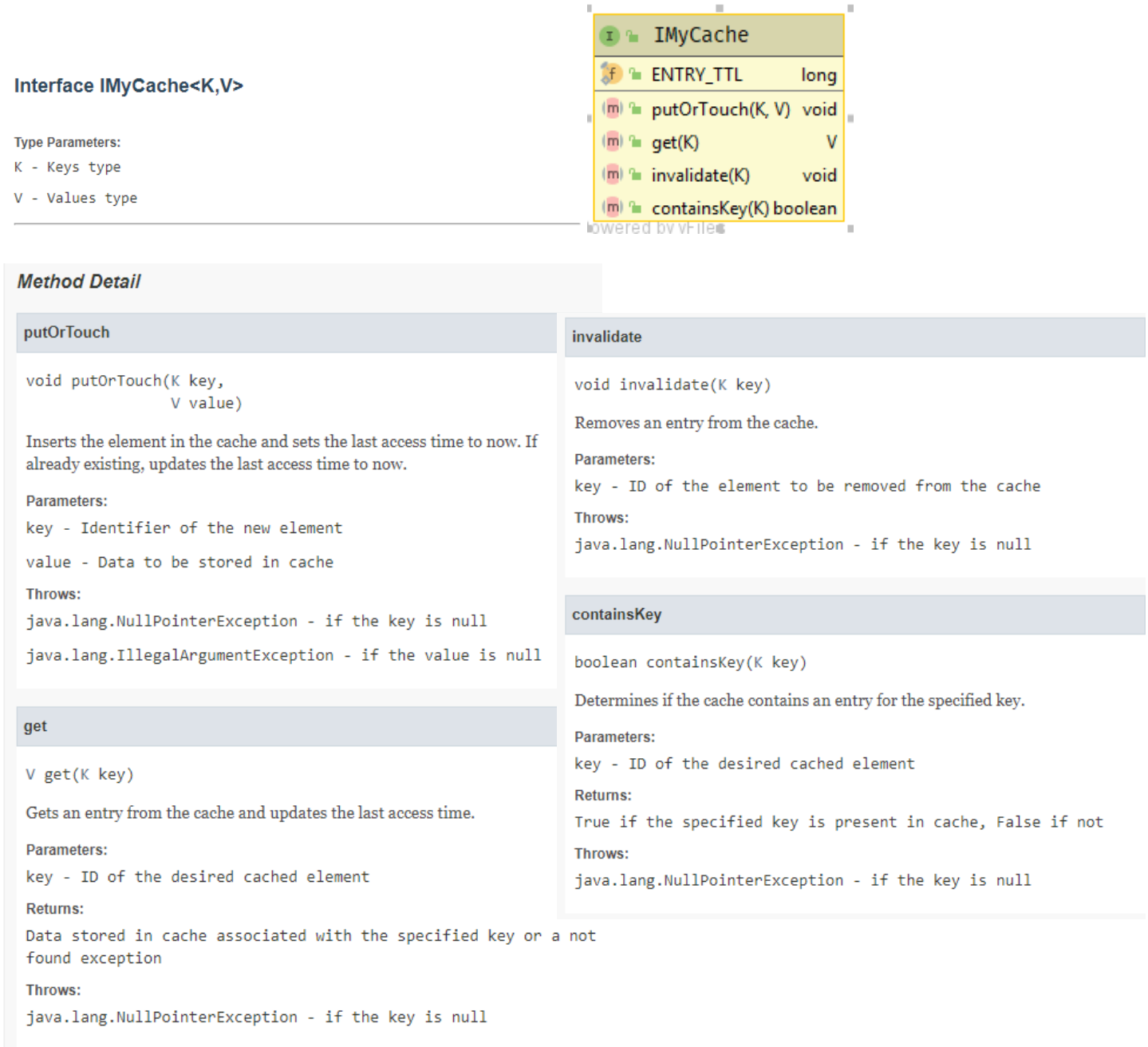

Figura 2: Interface para um módulo de Cache.