Época normal

1. **O modelo de qualidade do ISO-25010 identifica vários parâmetros relativos à qualidade do produto (de software), entre eles a "Maintainability, que está relacionado com:**

a) Integridade dos dados, não repúdio das ações, reutilização dos módulos. **- Modularity**
b) Facilidade de operação por parte dos utilizadores, tolerância a falhas pontuais do hardware, facilidade de alterar um componente sem introduzir defeitos ou degradar a qualidade já existente.
c) **Capacidade para estabelecer e executar testes de um sistema, uso de módulos para gerir a interdependência de componentes, capacidade de analisar o impacto de uma alteração no sistema global.**
d) Facilidade de testar, facilidade de operação, perceção da utilidade do sistema pelos utilizadores.
e) Eficiência na utilização interna dos recursos, tempos de resposta das operações de acordo com os requisitos estabelecidos, existência de mecanismos de prevenção dos erros de utilização.

2. **A metáfora da "pirâmide dos testes" quer transmitir a ideia de que:**

a) Existem testes mais importantes que outros (do topo para a base da pirâmide). **- todos os testes são importantes**
b) O esforço da equipa com as atividades de teste é cumulativo e aumenta de iteração para iteração. **- o esforço não aumenta necessariamente**
c) O número de testes diminui com o burndown, i.e., à medida que menos itens de trabalho subsistem no backlog, há menos testes para executar. **- Os testes executam-se na mesma**
d) **Os testes podem ser agrupados tendo em conta a sua granularidade e objetivos.**
e) Os testes das camadas superiores devem usar os testes das camadas inferiores. **- Testes devem ser independentes entre eles**

3. **Que problemas poderiam ser apontados a um portfólio de testes baseado numa "pirâmide invertida" (menos testes unitários, mais testes de aceitação)?**

a) **Demora muito tempo a executar, é difícil de manter, perde especificidade na localização dos problemas.**
b) Não é escalável porque os testes de aceitação requerem operação manual **- UAT também são exec automaticamente**

c) Obriga a fazer a gestão explícita de user stories, numa ferramenta adicional, não integradas no repositório de código. **- Nada a ver**
d) Não é adequado para projetos orientados à disponibilização de API/serviços. **- Não necessariamente**
e) É impeditivo da adoção de práticas de refactoring de código, pois não oferece a necessária "rede de segurança" para prevenir regressões. **- Nope**

4. **A análise estática de código:**

a) **é uma forma de detectar defeitos com muito baixo custo.**
b) permite a validação antecipada/precoce dos requisitos dos utilizadores. **- Não tem a ver c os requisitos**
c) a análise estática, com ferramentas completas, tornam o teste dinâmico desnecessário. **- Não**
d) a análise estática torna possível encontrar defeitos de runtime logo início do ciclo de desenvolvimento. **- Não são erros de runtime necesariamente**
e) na análise de vulnerabilidades de segurança, a análise estática tem menos valor que os testes dinâmicos, já que estes são mais eficazes a localizar os defeitos do código. **- Não**

5. **Qual das seguintes opções é a descrição mais adequada do conceito de cobertura (de instruções) de código?**

a) é uma métrica usada para medir a percentagem de testes que foram executados com sucesso.
b) é uma métrica que traduz o número de defeitos corrigidos sobre o total de defeitos encontrados.
c) é uma métrica utilizada para medir a percentagem de linhas de código fonte que são realmente executadas.
d) **é uma métrica que determina a relação das instruções que foram executadas num conjunto de testes, em relação ao total de instruções.**
e) é uma métrica que dá uma confirmação verdadeiro/falso se todas as instruções são utilizadas pelos testes.

6. **Nos projetos baseados em práticas ágeis, é mais necessário implementar a automação de testes do que em projetos "tradicionais", porque:**

Indicar quais as falsas e as verdadeiras
i. as alterações aos requisitos acontecem diariamente e os incrementos precisam de ser testados quanto a possíveis regressões. A alteração diária requer testes automatizados, porque o teste manual é muito lento. **- diariamente?**

ii. **os testes devem gerar feedback sobre a qualidade do produto o mais cedo possível. Portanto, todos os testes de aceitação devem ser executados em cada iteração, idealmente logo que as modificações são feitas, recorrendo a testes automatizados.**

iii. **a prática de integração contínua exige que o conjunto de testes de regressão seja executado sempre que o código é entregue, para gerar feedback sobre o estado da build. Na prática, só pode ser realizado por testes automatizados.**

iv. **As iterações/sprints são de duração fixa. A equipa deve garantir que todos os testes podem ser completamente executados no último dia de cada iteração/Sprint. Na prática, isso requer testes automatizados.**

v. Os projetos ágeis dependem de testes unitários em detrimento de testes de sistemas. Como os testes unitários não podem ser executados manualmente, todos os testes devem ser automatizados.

7. **Qual a definição mais adequada de "user story", tal como é usada nos testes em métodos ágeis?**

a) Um artefato a preparar pelos os testers, para detalhar apenas os requisitos funcionais do sistema. **- Não são os testers que escrevem**

b) Um artefacto do projeto, a produzir pelos programadores e que o Product Owner deve aprovar antes que os testes possa começar. **- Não são os programadores que escrevem**

c) Um artefato preparado pelos representantes do negócio/cliente para orientar os programadores e testers quanto às condições de aceitação do incremento. **- Não é o cliente que escreve**

d) Um artefato escrito colaborativamente pelos programadors, testers e especialistas do negócio para fixar os requisitos do produto **- É a equipa que escreve**

e) Um artefacto, apresentado de forma breve, para documentar novas funcionalidades e bugs que precisam de ser corrigidos.

8. **Segundo M. Fowler, qual das seguintes práticas NÃO é recomendada num sistema de integração contínua:**

a) As builds que falham devem ser corrigidas de imediato. **- "Fix problems builds immediately"**

b) **A realização de uma build deve ser rápida e, por isso, excluir testes de aceitação. - "Keep the builds fast".**

c) Os testes devem ser feitos em ambientes específicos, que "clonam" as condições de produção. **- "Test in a clone."**

d) Todos os programadores devem fazer entrega de código para o repositório partilhado, com regularidade (e.g.: pelo menos, diariamente). **- "Everyone commits daily".**

e) Todos os membros da equipa têm acesso imediato ao feedback do estado das builds. **- "Everyone can see what's happening".**

9. **O que é o pipeline de entrega contínua (continuous delivery)?**

a) É um ficheiro de configuração que deve acompanhar o repositório Git, de modo a que as novas entregas sejam detetadas de imediato.

b) **É uma implementação automatizada do processo de compilação, montagem, teste e instalação de uma aplicação.**

c) É um processo automático para instalar uma aplicação num ambiente de Cloud (e.g.: usando containers).

d) É uma visualização possível da execução dos projetos (jobs) configurados no Jenkins, considerando diferentes etapas na construção (stages), que dependentes do sucesso das antecedentes.

e) É a configuração das regras de qualidade (do código) e do nível de cobertura necessários para que a build passe com sucesso.

10. **Qual a hierarquia de elementos necessária na escrita de um pipeline declarativo do Jenkins, com inclusão de testes unitários?**

a) **Pipeline, agent, stages, stage, steps**

b) Node, agent, stages, stage, steps, step

c) Pipeline, docker, stages, stage, post

d) Pipeline, agent, stage, junit, post

e) Node, stage, test, deploy

pipeline { agent { stages { stage { steps { }}}

11. **Considere a implementação da story de login, com sucesso, quando o utilizador já se encontra registado. O interface baseado numa página web e acede aos serviços de retaguarda, invocando uma API. Que testes podem ser úteis, neste contexto?**

a) Testes unitários, para validar se a interação com a página web envolvida. **- Unit nao validam interações**

b) Testes de regressão, para considerar os dados de autenticação das tentativas de acesso anteriores e encontrar vulnerabilidades.

c) Testes de sistema, para confirmar o comportamento do serviço de autenticação, em situações de utilização intensiva. **- Para isso seriam testes de performance**

d) Testes de aceitação, para confirmar a conformidade das passwords com as regras de complexidade mínima definidas. **- UAT seriam para validar o UI**

e) **Teste unitário, para verificar o contrato dos métodos de cifra e validação de passwords.**

12. **Considere o quadro de apoio à decisão da Figura 1. Tendo em conta os compromissos que o gestor da qualidade deve praticar, considera que o projeto avaliado pode seguir para produção?**

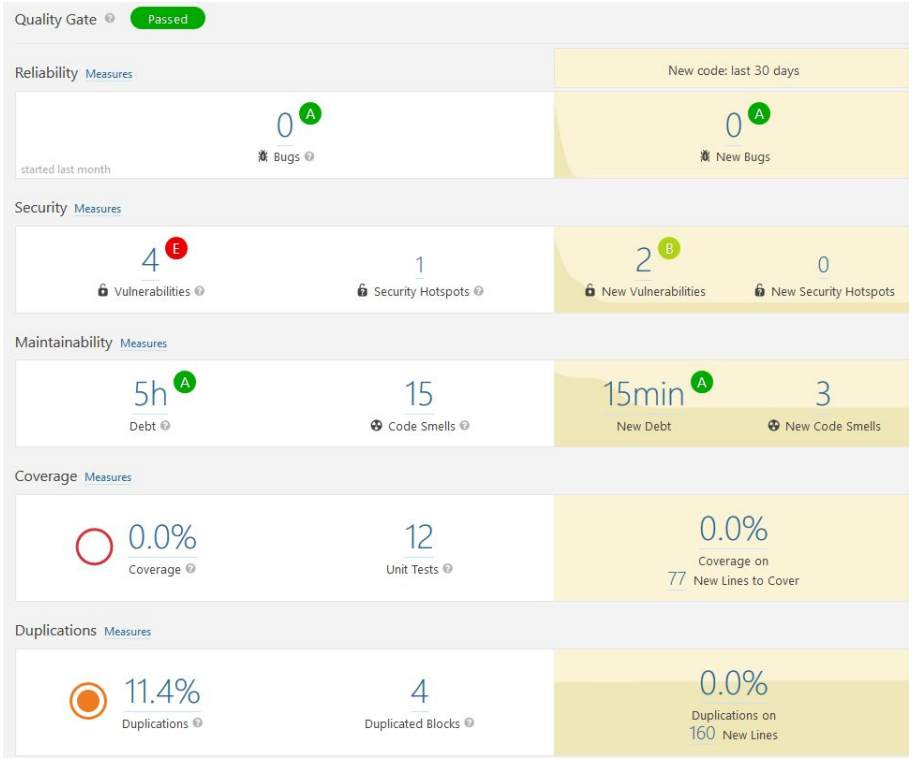
a) Sim. O "quality gate" definido (por omissão) está a passar e isso indica que o código não oferece problemas. **- ofc q tem**

b) Sim. Os indicadores de maintainability são claramente positivos. **- Não é o único fator que interessa**

c) Sim. Apesar de existirem 15 ocorrências construções que levantam dúvidas, podem ser resolvidas em 15 minutos. **- Tem 15 smells**

d) **Não. Existem vulnerabilidades de segurança graves e ainda não resolvidas.**

e) Não. Existe código duplicado em 4 blocos e tem de ser resolvido para melhorar a maintainability do produto. **- Não necessariamente a maintainability**



13. **Os testes funcionais implementados com recurso a Selenium/WebDriver podem recorrer ao padrão "page object model" (POM). Qual das seguintes afirmação NÃO é um benefício aplicável à utilização desse padrão?**

a) O padrão POM permite escrever testes funcionais muito mais legíveis (do que na utilização "normal" dos scripts de teste com Selenium). **- True**

b) O padrão POM permite reduzir a duplicação de código e separar a navegação (entre páginas) da verificação. **- True**

c) Os testes ficam mais focados e curtos, já que podemos reutilizar classes (que representam páginas) em diferentes testes.

d) As alterações no UI são facilmente implementadas, pois a manutenção necessária dos testes está bem compartimentada no modelo (cada página tem a sua classe associada).

e) **o padrão POM é diretamente suportado pelos métodos do WebDriver, que automatiza a instanciação dos objetos ("page objects"), à medida que são necessários no teste.**

14. **Quais as regras que deve observar uma equipa que adotou o "GitHub flow", na utilização do repositório partilhado?**

a) Trabalhar sobre o branch "master", com commits frequentes; manter um branch adicional para registar as releases do produto. **- Don't work on master**

b) Criar um novo branch para cada programador; fazer alterações no seu branch "privado"; integrar os incrementos no master partilhado. **- Nope**

c) **Criar um branch para cada nova feature; fazer alterações neste branch, com commits regulares; integrar o incremento no master, mediante um pedido de integração ("pull request").**

d) Criar um novo branch por feature; desenvolver a feature no respetivo branch; integrar no master; manter um branch adicional para as releases em separado.

e) Criar um novo branch local para cada feature; adicionar as alterações; integrar no master partilhado para revisão pelos pares. **- Not just local**

15. **Considere o trecho de código apresentado na Figura 2. Qual das seguintes práticas, aceitáveis para o tratamento de exceções, pode ser observada:**

a) **Não ignorar/suprimir as exceções.**

b) Evitar a utilização da exceções genéricas, usando blocos "try...catch" adequados. **- Tá a usar catch**

c) Adicionar contexto à exceção e lançar as exceções no nível de abstração adequado.

d) Registar a condição de exceção num "log" da aplicação.

e) Tratar a exceção no nível adequado da hierarquia de invocação dos objetos.

```
6   public void queryCustomer(javax.servlet.ServletRequest request) {
7       try {
8           Connection connection = DriverManager
9                   .getConnection( url: "jdbc:mysql://localhost:3306/JDBCDemo",
10                          user: "root",  password: "secret");
11
12          String query = "SELECT account_balance FROM user_data WHERE user_name = "
13                  + request.getParameter( s: "customerName");
14
15          Statement statement = connection.createStatement();
16          ResultSet results = statement.executeQuery(query);
17
18          doSomethingWithTheseResults( results);
19
20      } catch (Exception e) {
21          e.printStackTrace();
22      }
23  }
```

16. Qual o ciclo característico de uma abordagem TDD?

a) Adicionar o novo incremento; escrever os testes unitários que o verificam; executar todos os testes; melhorar o código, se necessário.

b) Adicionar os testes relativos à user story em mãos; executar os testes e confirmar que falham; implementar o código necessário para fazer passar os testes; rever e aceitar o incremento.

c) Adicionar um teste; executar todos os testes e ver o novo a falhar; fazer as alterações necessárias para o teste passar; correr todos os testes e confirmar que passam; rever o código (refactoring).

d) Limpar o código anterior; adicionar um novo teste; implementar o código necessário e submeter no repositório partilhado; observar o feedback do sistema de CI.

e) Escrever os testes unitários no início da interação; implementar o código necessário para fazer passar os testes; escrever os testes de integração; fazer refactoring do código na medida necessária.

17. A existência de código duplicado indica, geralmente, uma má prática. Que refactoring INADEQUADO para resolver este code smell?

a) os blocos de código duplicados ocorrem dentro da mesma classe: introduzir uma constante e atribuir-lhe o bloco de código comum.

b) o mesmo código é encontrado em duas subclasses do mesmo nível: extrair para um método, e colocá-lo num nível de hierarquia acima. **- Adequado**

c) o mesmo código é encontrado em duas subclasses do mesmo nível: se o código duplicado estiver dentro de um construtor, colocar a parte comum num construtor num nível acima. **- Adequado**

d) se o código duplicado for encontrado em duas classes diferentes, extrair o bloco para uma superclasse nova e as classes mantém as funcionalidades anteriores. **- Adequado**

e) se o código duplicado for encontrado em duas classes diferentes, mas não for natural criar uma única superclasse, criar uma classe adicional e mover para lá o bloco comum. **- Adequado**

18. **A utilização de ambientes de mocking ajuda na utilização de objetos sintetizados, em substituição de objetos/serviços reais. Qual das afirmações NÃO é uma vantagem atribuível a estes ambientes?**

a) Capacidade de retornar valores extremos, para exercitar condições limite. **- Vantagem**

b) Manter os testes unitários rápidos, isolados de potenciais latências (de serviços necessários). **- Vantagem**

c) Introduzir previsibilidade no resposta de um serviço remoto;

d) **Fornecer uma implementação simplificada de um módulo atribuído a outro programador;** **- Stubs**

e) Facilitar a preparação das condições necessárias para o teste, através da definição das expectativas. **- Vantagem**

19. **Os testes unitários são escritos pelo programador; NÃO são úteis para o ajudar a:**

a) entender o contrato do módulo (requisitos do que vai construir);

b) **escrever menos código;**

c) documentar a utilização pretendida de um componente;

d) prevenir erros de regressão;

e) aumentar a confiança no código.

20. **Qual a interpretação mais adequada da conceito "dívida técnica" (technical debt)?**

a) É a diferença do nível de cobertura atual para os 100% de cobertura.

b) É o número de erros encontrados num projeto.

c) É o número de user stories não aceites na iteração corrente, por não passarem os testes definidos.

d) **É uma estimativa do tempo necessário para corrigir os problemas encontrados durante a análise estática.**

e) É uma estimativa do tempo de trabalho necessário para fazer passar os testes que estão a falhar.

Época Recurso

1. **Qual o propósito da utilização de um modelo de qualidade (como o ISO-25010)?**

A. Garantir a integridade dos dados, o não repúdio das ações e a reutilização dos módulos.
B. Avaliar o grau até onde o conjunto de funcionalidades implementadas cobre a totalidade das necessidades e objetivos dos utilizadores.
C. Determinar um conjunto de características de referência que devem ser consideradas quando se avalia as propriedades de um produto de software.
D. Facilitar a integração (compatibilidade) entre sistemas, especialmente quando se trata de um modelo normalizado (e.g.: norma ISO).
E. Avaliar a facilidade com que os utilizadores de um sistema aprendem a usá-lo de forma eficiente e segura.

C)

2. **A metáfora da "pirâmide dos testes" (de Cohen) transmite a ideia de que:**

A. O esforço da equipa com as atividades de teste é cumulativo e aumenta de iteração para iteração.
B. O número de testes diminui com o *burndown*, i.e., à medida que menos itens de trabalho subsistem no *backlog*, há menos testes para executar.
C. Os testes das camadas superiores devem usar os testes das camadas inferiores.
D. Existem diferentes classes de teste de software, que variam em quantidade e quanto aos seus objetivos.
E. Nas metodologias ágeis (associadas à metáfora da "pirâmide") as práticas de teste são o inverso do modelo tradicional ("V-Model").

D)

3. **Se quiser defender a adoção de uma abordagem BDD numa equipa, qual dos argumentos NÃO É correto?**

A. Contribui para colocar o foco do teste nas necessidades do utilizador e não nos detalhes técnicos da implementação.
B. É suportada por uma linguagem natural que permite descrever o teste na área do domínio do problema, entendida pela equipa alargada.
C. 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.
D. Facilita a compreensão do comportamento esperado do software, ao fomentar a colaboração entre a equipa técnica e a equipa do "negócio".
E. Os testes unitários são executáveis e substituem a necessidade de criar especificações de requisitos adicionais.

E) - Não substituem a necessidade

4. **Uma estratégia de garantia de qualidade deve incluir testes de caixa-aberta e testes de caixa-fechada.**

A. A análise estática de código é um exemplo de uma estratégia de testes de caixa-aberta.
B. Os testes unitários são testes de caixa-aberta que exercitam os caminhos internos de um módulo.
C. O nível inferior da "pirâmide de testes" é genericamente caraterizado por testes de caixa-fechada, enquanto que o nível superior por testes de caixa-aberta.
D. Os testes funcionais são testes de caixa-fechada, eficazes para localizar a origem dos erros no código.
E. O conhecimento da implementação interna é fundamental para escrever testes funcionais eficientes.

A) - Caixa aberta - Test the implementation
B) - Caixa fechada - Test the functionality
C) - É o oposto
D) - São de caixa aberta
E) - São de caixa fechada

5. **O nível de cobertura reportado no SonarQube para o Projeto XYZ é de 70%. Isso significa que:**

A. A dívida técnica do projeto é de 30%.
B. Não se deve avançar para a produção; é altamente desejável obter um nível de cobertura de 100%.
C. A maior parte do código do projeto foi exercitado nos testes disponíveis, mesmo que repetidamente.
D. 70% dos testes executados passaram, e 30% não estão a passar.
E. A análise estática determinou que a probabilidade de não existirem problemas no código é de 70%.

C)

6. **O que é a "user story", tal como é usada nos processos de Garantia de Qualidade em métodos ágeis?**

A. Artefacto usado para monitorizar o progresso, identificado e priorizado por equipas interdisciplinares.
B. Artefacto preparado pelos *testers*, para detalhar os requisitos funcionais do sistema e os critérios de aceitação.
C. Um artefacto do projeto, a produzir pelo *Product Owner* para fazer a aceitação dos incrementos entregues pelos programadores.
D. Um artefacto preparado pelos representantes do negócio/cliente para apresentarem as histórias que pretendem realizar no sistema.
E. Um artefacto, apresentado de forma breve, para documentar novas funcionalidades a desenvolver na presente iteração/sprint.

A)

7. **Qual das seguintes práticas NÃO ESTÁ de acordo com as dez recomendadas por M. Fowler, relativamente à preparação de um sistema de integração contínua:**

A. As *builds* que falham devem ser corrigidas de imediato.
B. O projeto deve manter um repositório para *branches* de desenvolvimento e um repositório para integração (*master*).
C. Devem existir teste feitos em ambientes que mimetizam as condições de produção.
D. Todos os membros da equipa têm acesso imediato ao *feedback* do estado das *builds*.
E. Deve existir formas automáticas de fazer as instalações, para montar os ambientes de teste e, quando for o caso, de produção.

B) - Use a single repository

8. **A expressão "pipeline as code", associada, por exemplo, às versões mais recentes do Jenkins, significa que:**

A. Um *pipeline* de CI inclui a compilação de código fonte e produz artefactos são partilhados na equipa.
B. Os passos do *pipeline* são executados quando é entregue novo código fonte no repositório associado.
C. A definição do *pipeline* é escrita numa linguagem própria e o esse ficheiro sujeito ao controlo de versões, no repositório de código.
D. É possível executar condicionalmente algumas etapas do *pipeline*, fazendo-as depender do sucesso das antecedentes.
E. O *pipeline* é compilado, como o resto do código fonte, na realização das tarefas da ferramenta de montagem (e.g.: Maven).

C)

9. **Qual a hierarquia de elementos necessária na escrita de uma pipeline declarativo do Jenkins, com inclusão de testes de integração?**

A. Pipeline, docker, stages, stage, post
B. Pipeline, agent, stages, stage, steps
C. Pipeline, agent, stage, mock, post
D. Node, agent, stages, step, springboottest
E. Node, stage, test, deploy

B)

10. **Considere a história: "Sendo um viajante, QUERO reservar um quarto para ficar no período de tempo indicado, DE MODO A não me preocupar mais em procurar sítio". Que situação de teste NÃO É indicada/relevante para a validação da funcionalidade?**

A. Testes unitários, para verificar o comportamento da operação de reserva do módulo gestor de reservas.
B. Testes funcionais, para confirmar que o viajante consegue, na página web, selecionar as datas e inserir a reserva.
C. Testes de aceitação, para confirmar que as mensagens presentes ao Viajante, na página, são claras e informativas.
D. Testes de aceitação, para confirmar que o Viajante obtém *feedback* claro e informativo, no caso de a marcação não poder ser satisfeita.
E. Testes de regressão, para verificar que as novas reservas não colidem com reservas já existentes.

E)

11. Considere a informação da Figura 2. Assumindo Assumindo que a solução assume as práticas comuns de desenvolvimento e teste com Spring Boot, assinale a afirmação FALSA:

A. O componente sob teste é o RegisterRestController.
B. O componente RegisterUseCase não é usado no teste.
C. O teste "*whenValidInput...*" insere uma entidade e verifica que o serviço *registerUser* foi invocado exatamente uma vez.
D. Apesar de o teste usar o protocolo HTTP como consumidor de serviços, não inicia verdadeiramente um servidor web com a aplicação.
E. O teste verifica o (código de) estado e o conteúdo da resposta do pedido HTTP para avaliar o comportamento esperado do componente.

B)

```
21  @WebMvcTest(controllers = RegisterRestController.class)
22  class RegisterRestControllerTest {
23
24      @Autowired
25      private MockMvc mockMvc;
26
27      @Autowired
28      private ObjectMapper objectMapper;
29
30      @MockBean
31      private RegisterUseCase registerUseCase;
32
33      @Test
34      void whenValidInput_thenMapsToBusinessModel() throws Exception {
35
36          UserResource user = new UserResource( name: "Zaphod",  email: "zaphod@galaxy.net");
37
38          mockMvc.perform(post( urlTemplate: "/forums/{forumId}/register",  ...uriVars: 42L)
39                  .contentType("application/json")
40                  .param( name: "sendWelcomeMail",  ...values: "true")
41                  .content(objectMapper.writeValueAsString(user)))
42                  .andExpect(status().isOk());
43
44          ArgumentCaptor<User> userCaptor = ArgumentCaptor.forClass(User.class);
45          verify(registerUseCase, times( wantedNumberOfInvocations: 1)).registerUser(userCaptor.capture(),
46                  eq( value: true));
47          assertThat(userCaptor.getValue().getName()).isEqualTo("Zaphod");
48          assertThat(userCaptor.getValue().getEmail()).isEqualTo("zaphod@galaxy.net");
49
50      }
```

12. Os testes funcionais implementados com recurso a Selenium/WebDriver podem recorrer ao padrão "page object model" (POM). Em que consiste este padrão?

A. A ferramenta Maven utiliza a modelo do POM para diferenciar entre testes unitários e de integração, executado os testes "Selenium" neste último grupo.
B. O padrão POM utiliza exemplos escritos em linguagem do domínio (*features*) para alimentar a execução dos "testes Selenium".
C. Cada página web tem um controlador (da interação) associado, sendo possível mapear os campos na página com os atributos do controlador.
D. A lógica de interação sob uma página, através de WebDriver, deve ficar estruturada em métodos de uma classe, que representa essa página nos testes.
E. A programação da página web deve usar uma linguagem por objetos e um modelo de interação MVC, facilitadora dos testes.

D)

13. Quais as regras que deve observar uma equipa que adotou o "GitHub Flow", na utilização do repositório partilhado?

A. Trabalhar sobre o *branch* "master", com *commits* frequentes; manter um *branch* adicional para registar as *releases* do produto.
B. Criar um novo *branch* para cada programador; fazer alterações no seu *branch* "privado"; integrar os incrementos no *master* partilhado.
C. Criar um novo *branch* local para cada *feature*; adicionar as alterações; integrar no *master* partilhado para revisão pelos pares.
D. Criar um novo *branch* por *feature*; desenvolver a *feature* no respetivo *branch*; integrar no *master*; manter um *branch* adicional para as *releases* em separado.
E. Criar um *branch* para cada nova *feature*; fazer alterações neste *branch*, com *commits* regulares; integrar o incremento no *master*, mediante um pedido de integração ("*pull request*").

E)

14. Considere o trecho de código apresentado na Figura 2. Considere que o teste é executado sem erros, mas que falha (i.e., o teste corre, mas não passa). Que instruções podem sinalizar ao framework de teste que as condições exercitadas não "passam"?

A. As instruções das linhas 47 e 48.
B. As instruções das linhas 45, 47 e 48;
C. As instruções das linhas 25, 28, 31, 47 e 48.
D. As instruções das linhas 38, 45, 47 e 48.
E. As instruções das linhas 31, 45, 47 e 48.

D)

```
21  @WebMvcTest(controllers = RegisterRestController.class)
22  class RegisterRestControllerTest {
23
24      @Autowired
25      private MockMvc mockMvc;
26
27      @Autowired
28      private ObjectMapper objectMapper;
29
30      @MockBean
31      private RegisterUseCase registerUseCase;
32
33      @Test
34      void whenValidInput_thenMapsToBusinessModel() throws Exception {
35
36          UserResource user = new UserResource( name: "Zaphod",  email: "zaphod@galaxy.net");
37
38          mockMvc.perform(post( urlTemplate: "/forums/{forumId}/register",  ...uriVars: 42L)
39                  .contentType("application/json")
40                  .param( name: "sendWelcomeMail",  ...values: "true")
41                  .content(objectMapper.writeValueAsString(user)))
42                  .andExpect(status().isOk());
43
44          ArgumentCaptor<User> userCaptor = ArgumentCaptor.forClass(User.class);
45          verify(registerUseCase, times( wantedNumberOfInvocations: 1)).registerUser(userCaptor.capture(),
46                  eq( value: true));
47          assertThat(userCaptor.getValue().getName()).isEqualTo("Zaphod");
48          assertThat(userCaptor.getValue().getEmail()).isEqualTo("zaphod@galaxy.net");
49
50      }
```

15. Qual o ciclo característico de uma abordagem BDD?

A. Adicionar o novo incremento; escrever os testes funcionais que o verificam; executar todos os testes; melhorar o código, se necessário.
B. Adicionar um teste; executar todos os testes e ver o novo a falhar; fazer as alterações necessárias para o teste passar; correr todos os testes e confirmar que passam; rever o código (*refactoring*).
C. Descrever o comportamento esperado com exemplos; escrever um teste com os passos correspondentes aos do exemplo; verificar que falha; fazer as alterações para o teste passar; reformular o código (*refactoring*).
D. Escrever uma história (*user story*); adicionar um novo teste; implementar o código da funcionalidade; observar o *feedback* do sistema de CI.
E. Escrever os testes no início da iteração; verificar que estão a falhar; implementar o código necessário para fazer passar os testes; fazer *refactoring* do código na medida necessária.

C)

16. A utilização de ambientes de mocking ajuda na utilização de objetos sintetizados, em substituição de objetos/serviços reais. Qual das afirmações NÃO É uma vantagem atribuível a estes ambientes?

A. Conveniência para retornar valores limite dos parâmetros para exercitar diferentes percursos no módulo sob teste.
B. Manter os testes unitários rápidos, isolados de potenciais latências (de serviços necessários).
C. Introduzir previsibilidade na resposta de um serviço remoto;
D. Facilitar a preparação das condições necessárias para o teste, através da definição das espectativas.
E. Contar o número de vezes que o teste é executado (com cláusulas de verificação).

E)

17. **Uma utilização natural do framework Mockito no teste de um componente SpringBoot (SB) anotado com o @Service, seria:**

A. Sintetizar um objeto do tipo repositório, substituindo o acesso concreto à base de dados, por respostas predefinidas.
B. Substituir o contexto do servidor web por uma versão mais simples no teste, acelerando a execução.
C. Viabilizar a injeção das dependências necessárias para realizar o teste, automaticamente satisfeitas pelo SB.
D. Avaliar a resposta dos métodos da API REST sob teste, por exemplo, quanto ao código de retorno bem-sucedido.
E. Viabilizar testes de integração, em conjunto com a anotação @SpringBootTest, executados num contexto da aplicação especial para os testes.

A) - @Mock
D) - @InjectBeans, logo ñ é feito automaticamente pelo SB

18. **Os testes unitários são uma peça importante numa estratégia de garantia de qualidade, no entanto, também se podem observar falácias (pitfalls) na sua utilização, tais como:**

A. A escrita de testes unitários à cabeça ajuda a entender o contrato do módulo que se vai construir;
B. Os testes unitários ajudam a revelar erros de regressão no novo código;
C. Quanto maior o número de testes unitários, mais confiança podemos ter na *build*.
D. Contribuem para documentar a utilização esperada de um componente, facilitando a manutenção;
E. Aumentam a confiança da equipa para executar operações de reformulação do código (*refactoring*) com frequência.

C)

19. **Qual a interpretação mais adequada do conceito "dívida técnica" (technical debt), usado em ambientes de qualidade?**

A. É a diferença do nível de cobertura atual para os 100% de cobertura.
B. É a diferença da velocidade atual da equipa (medida como a média de pontos aceites nas últimas 3 iterações) para a velocidade estabelecida como objetivo.
C. É o número de *user stories* não aceites na iteração corrente, por não passarem os testes definidos.

D. É uma estimativa do tempo de trabalho necessário para fazer passar os testes que estão a falhar.
E. É uma estimativa do tempo necessário para corrigir os problemas encontrados durante a análise estática.

E)

20. Relativamente à utilização da exceção NullPointerException nos contratos dos métodos da Figura 3:

A. É adequada, visto que os utilizadores do módulo (i.e., programadores) devem assegurar a verificação dos parâmetros antes de chamar os métodos.
B. É desadequada, porque não dá ao programador que usa o módulo a possibilidade de receber e tratar a condição de exceção.
C. É adequada, visto que as exceções "checked" são sempre incluídas na documentação do módulo, alertando o programador para essa eventualidade.
D. É desadequada, uma vez que não acrescenta valor ao módulo, já que na impossibilidade de retornar valores encontrados, os métodos podem retornar "null".
E. O módulo seria mais fácil de usar se os métodos lançassem exceções especializadas, com a semântica detalhada da condição verificada (e.g.: KeyNotFoundException)

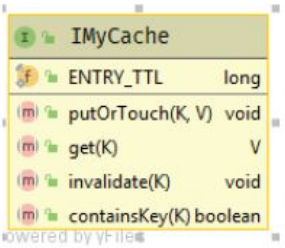
E)

Interface IMyCache<K,V>

Type Parameters:

K - Keys type

V - Values type



Method Detail

putOrTouch

```
void putOrTouch(K key,
                V value)
```

Inserts the element in the cache and sets the last access time to now. If already existing, updates the last access time to now.

Parameters:

key - Identifier of the new element

value - Data to be stored in cache

Throws:

java.lang.NullPointerException - if the key is null

java.lang.IllegalArgumentException - if the value is null

get

```
V get(K key)
```

Gets an entry from the cache and updates the last access time.

Parameters:

key - ID of the desired cached element

Returns:

Data stored in cache associated with the specified key or a not found exception

Throws:

java.lang.NullPointerException - if the key is null

invalidate

```
void invalidate(K key)
```

Removes an entry from the cache.

Parameters:

key - ID of the element to be removed from the cache

Throws:

java.lang.NullPointerException - if the key is null

containsKey

```
boolean containsKey(K key)
```

Determines if the cache contains an entry for the specified key.

Parameters:

key - ID of the desired cached element

Returns:

True if the specified key is present in cache, False if not

Throws:

java.lang.NullPointerException - if the key is null