

**Engenharia de Software - Trabalho Prático (30% nota)**

**2º ano – Tecnologias de Informação**

**Relatório**

Gonçalo Farias -109180

Índice

**Introdução1**

Descrição Detalhada do Problema

Objetivo do Teste

**Estratégias de Teste2**

Testes Unitários

Estratégias Abordadas

**Justificação de Casos de Teste3**

**Partição do Domínio de Entrada4**

**Modelo do Grafo5**

**Conclusão6**

1. **Introdução**

O desafio proposto, conhecido como “PimPamPum”, envolve a implementação de um programa em Java que seja capaz de analisar um número fornecido como argumento e modificar a sua representação com base em diferentes condições. Este problema exige uma abordagem criteriosa, considerando a divisibilidade pelos números (3, 5 e 7) e a presença dos dígitos (3, 5 e 7) no número em questão.

A essência da questão é criar um código limpo, eficiente e claro, que siga princípios ágeis que priorizem a simplicidade e a compreensão do código. É utilizada então uma abordagem de desenvolvimento orientada a testes unitários para garantir a qualidade do código e facilitar modificações futuras.

* **Descrição Detalhada do Problema**

O objetivo consiste em criar um programa que após receber um número como argumento, faça as seguintes alterações:

* Número divisível por 3, substituir por “Pim”.
* Número divisível por 5, substituir por “Pam”.
* Número divisível por 7, substituir por “Pum”.
* Dígito 3 contido no número, substituir por “Pim”.
* Dígito 5 contido no número, substituir por “Pam”.
* Dígito 7 contido no número, substituir por “Pum”.

É importante ir abordando o problema de uma forma modular e incremental. Tendo sido divido o programa em duas partes, uma onde se trata a divisibilidade do número e a outra onde trata o comportamento de dígitos específicos.

* **Objetivo do Teste**

O objetivo principal dos testes feitos por mim é assegurar que a implementação do método “run” na classe “PimPamPum” esteja correta e atenda sempre aos requisitos que foram estabelecidos no âmbito do projeto. Estes testes validam o programa através de diversos tipos de cenários que podem acontecer, assegurando que as condições impostas sejam respeitadas.

1. **Estratégias de Teste**

* **Testes Unitários**

A estratégia de utilizar os testes unitários, teve como base o facto de pretender garantir que cada unidade do código fosse testada de forma isolada e individual, facilitando a identificação de falhas em certas partes do código.

* **Estratégias Abordadas**

**1.** Testar se o método retorna a palavra “Pim” para números divisíveis por 3;

**2.** Testar se o método retorna a palavra “Pam” para números divisíveis por 5;

**3.** Testar se o método retorna a palavra “Pum” para números divisíveis por 7;

**4.** Testar se o método retorna corretamente as palavras “Pim” e “Pam” para números que contenham os dígitos 3 e 5, respetivamente;

**5.** Testar se o método retorna corretamente as palavras “Pim”, “Pam” e “Pum” para números que contenham os dígitos 3, 5 e 7, respetivamente;

**6.** Testar se o método retorna o próprio número quando ele não é divisível por nenhum dos valores (3, 5 ou 7) e não contém os dígitos 3, 5 ou 7.

1. **Justificação de Casos de Teste**

Foram desenvolvidos casos de teste específicos para garantir uma cobertura abrangente, incluindo:

1. **testRun\_DivisivelPor3:** Este teste tem como objetivo verificar se o método retorna corretamente a palavra “Pim” para números divisíveis por 3.
2. **testRun\_DivisivelPor5:** O objetivo deste teste é verificar se o método retorna corretamente a palavra “Pam” para números divisíveis por 5.
3. **testRun\_DivisivelPor7:** O objetivo deste teste é verificar se o método retorna corretamente a palavra “Pum” para números divisíveis por 7.
4. **testRun\_Conter3:** O objetivo deste teste é verificar se o método retorna corretamente a palavra “Pim” para números que contêm o dígito 3.
5. **testRun\_Conter5:** O objetivo deste teste é verificar se o método retorna corretamente a palavra “Pam” para números que contêm o dígito 5.
6. **testRun\_Conter7:** O objetivo deste teste é verificar se o método retorna corretamente a palavra “Pum” para números que contêm o dígito 7.
7. **testRun\_DivisivelPor3Conter3:** O objetivo deste teste é verificar se o método retorna corretamente a palavra "Pim" para números que são divisíveis por 3 e também contêm o dígito 3.
8. **testRun\_DivisivelPor3Por5Conter5:** O objetivo deste teste é verificar se o método retorna corretamente as palavras “Pim” e “Pam” para números que são divisíveis por 3 e por 5, e também contêm o dígito 5.
9. **testRun\_DivisivelPor3Por7:** O objetivo deste teste é verificar se o método retorna corretamente as palavras “Pim” e “Pum” para números que são divisíveis por 3 e por 7.
10. **testRun\_Conter3Conter5:** O objetivo deste teste é verificar se o método retorna corretamente as palavras “Pam” e "Pim" para números que contêm os dígitos 3 e 5, respetivamente.
11. **testRun\_NaoDivisivelPor3Por5:** O objetivo deste teste é verificar que o método não retorna “Pim” nem “Pam” para números não divisíveis por 3 e 5.
12. **testRun\_DivisivelPor3Conter3Conter5Conter7:** O objetivo deste teste é verificar se o método retorna corretamente as palavras “Pim”, “Pum”, “Pam” e “Pim” para números que são divisíveis por 3, e também contêm os dígitos 3, 5 e 7.
13. **testRun\_NaoDivisivel:** O objetivo deste teste é verificar se o método retorna o próprio número quando ele não é divisível por nenhum dos valores (3, 5 ou 7) e não contém os dígitos 3, 5 ou 7. Isso é importante para garantir que o método esteja a tratar corretamente casos em que nenhuma das regras se aplica.
14. **testRun\_NaoConterDigitosIguais:** O objetivo deste teste é verificar se o método retorna o próprio número quando ele não contém nenhum dos dígitos 3, 5 ou 7. Isso é importante para garantir que o método esteja a tratar corretamente casos em que não há necessidade de substituição por palavras.
15. **testRun\_NumeroNegativo:** O objetivo deste teste é verificar se o método lança uma exceção ‘IllegalArgumentException’ quando o número passado como parâmetro é negativo. Isso é importante para garantir que o método esteja a tratar adequadamente casos inválidos.
16. **testRun\_NumeroZero:** O objetivo deste teste é verificar se o método retorna corretamente as palavras “Pim”, “Pam” e “Pum” para o número zero. Isso é importante para garantir que o método esteja a tratar adequadamente casos especiais, como zero.
17. **Partição do Domínio de Entrada**

A técnica de particionamento do domínio de entrada foi aplicada para garantir uma cobertura abrangente dos diferentes cenários que o código deve lidar. Abaixo estão as principais partições identificadas e os resultados obtidos nos testes correspondentes:

**Números Divisíveis por 3:**

Foram selecionados casos de teste representativos, como 9 e 33, para avaliar a correta identificação de números divisíveis por 3. Os resultados foram consistentes com as expectativas, com a execução bem-sucedida do cenário "Pim".

**Números Divisíveis por 5:**

Casos de teste, como 10 e 15, foram escolhidos para verificar a deteção adequada de números divisíveis por 5. Os resultados confirmaram o comportamento esperado, com a ativação do cenário "Pam".

**Números Divisíveis por 7:**

Testes foram conduzidos com números como 14 e 21 para assegurar que o código reconhece corretamente números divisíveis por 7. O cenário "Pum" foi acionado conforme previsto.

**Números que Contêm o Dígito 3:**

Foram selecionados números como 3 e 33 para avaliar a deteção de dígitos específicos. Os resultados demonstraram que o código executa a ação "Pim" quando o dígito 3 está presente.

**Números que Contêm o Dígito 5:**

Testes foram conduzidos com números como 5 e 15 para verificar a correta identificação de dígitos 5. Os resultados confirmaram que o código ativa o cenário "Pam" quando o dígito 5 está presente.

**Números que Contêm o Dígito 7:**

Casos de teste, como 7 e 753, foram utilizados para validar a deteção de dígitos 7. Os resultados indicaram que o código executa a ação "Pum" quando o dígito 7 está presente.

**Números que Não Atendem a Nenhuma Condição Específica:**

Testes foram realizados com números como 11 e 22, que não atendem a nenhuma condição específica. Os resultados confirmaram que o código mantém o número inalterado nessas situações.

**Números Negativos:**

Cenários com números negativos, como -5, foram testados para garantir que o código lide adequadamente com essas entradas. Os resultados demonstraram um tratamento apropriado.

**Número Zero:**

O número zero foi incluído nos testes para verificar se o código responde conforme o esperado. O resultado foi consistente, com a ação "PimPamPum" a ser executada.

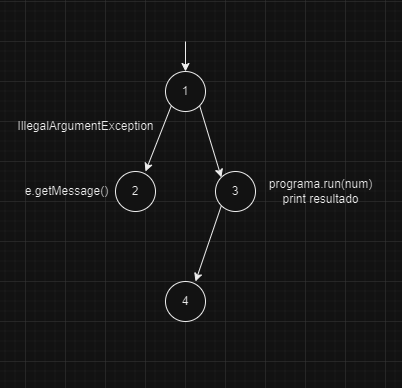
1. **Modelo do Grafo**

Nesta secção, apresento os modelos de grafo que desenvolvi para os métodos principais do projeto “PimPamPum”. Estes modelos proporcionam uma visão clara e concisa do fluxo de execução do programa, contribuindo para uma compreensão mais aprofundada do código.

**Método “main”:**

**Node Coverage:** TR = {1,2,3,4} Test Paths: [1,2] [1,3,4]

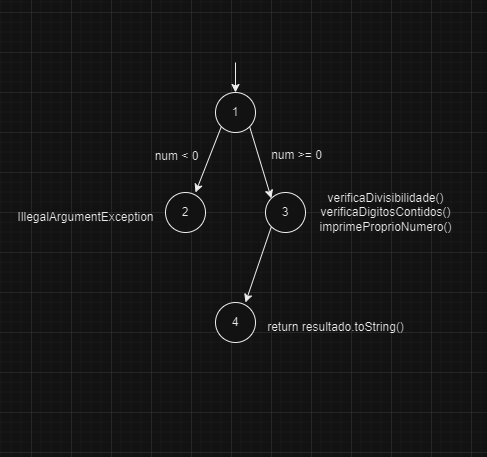
**Edge Coverage:** TR = {(1,2), (1,3), (3,4)} Test Paths: [1,2] [1,3,4]

****

**Método “run”:**

**Node Coverage:** TR = {1,2,3,4} Test Paths: [1,2] [1,3,4]

**Edge Coverage:** TR = {(1,2), (1,3), (3,4)} Test Paths: [1,2] [1,3,4]

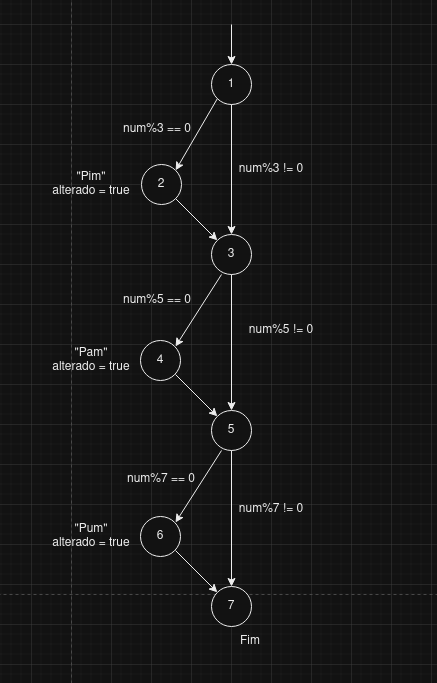
****

**Método “verificaDivisibilidade”:**

**Node Coverage:** TR = {1,2,3,4,5,6,7} Test Paths: [1,2] [1,3,4]

**Edge Coverage:** TR = {(1,2), (1,3), (2,3), (3,4), (3,5), (4,5), (5,6), (5,7), (6,7)}

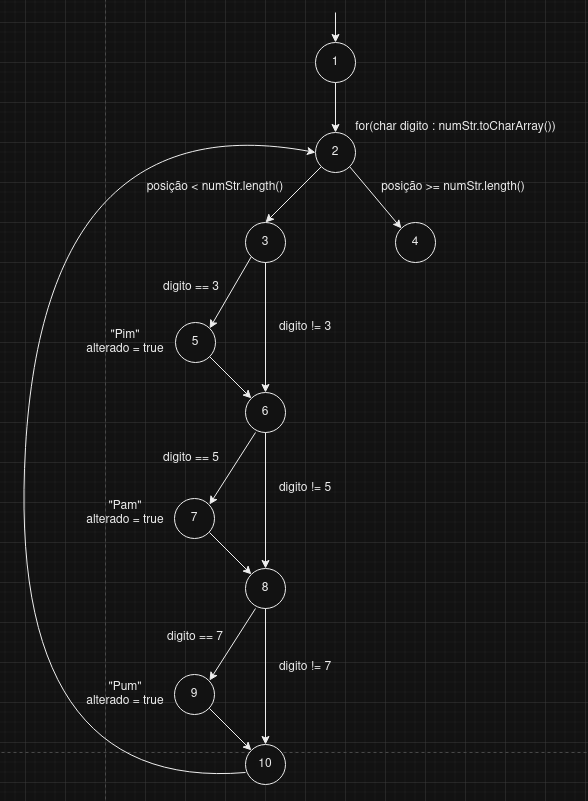
Test Paths: [1,2,3,4,5,6,7] [1,3,5,7]

****

**Método “verificaDigitosContidos”:**

**Node Coverage:** TR = {1,2,3,4,5,6,7,8,9,10} Test Paths: [1,2,4] [1,2,3,5,6,7,8,9,10]

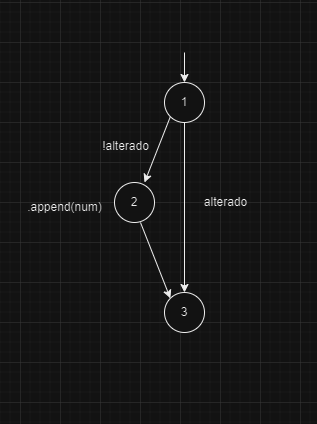
**Edge Coverage:** TR = {(1,2), (2,3), (2,4), (3,5), (3,6), (5,6), (6,7), (6,8), (7,8), (8,9), (8,10), (9,10)} Test Paths: [1,2,4] [1,2,3,5,6,7,8,9,10] [1,2,3,6,8,10]

****

**Método “imprimeProprioNumero”:**

**Node Coverage:** TR = {1,2,3} Test Paths: [1,2,3]

**Edge Coverage:** TR = {(1,2), (1,3), (2,3)} Test Paths: [1,2,3] [1,3]

****

**6. Conclusão**

As estratégias de teste adotadas e justificativas dos casos de teste implementados visam garantir que o método ‘run’ da classe ‘PimPamPum’ esteja a funcionar corretamente de acordo com as especificações do problema. Os casos de teste foram selecionados com base nas diferentes regras definidas no enunciado do problema e em possíveis cenários alternativos. Ao realizar os testes, foi possível garantir que todas as regras estão a ser aplicadas corretamente e que o método trata adequadamente possíveis casos excecionais.

A técnica de partição do domínio de entrada aprimorou a análise, proporcionando uma compreensão mais aprofunda do programa em diversos contextos. A identificação e teste de categorias específicas, como números divisíveis por 3, 5 ou 7, e aqueles contendo os dígitos 3, 5 ou 7, validaram integralmente o código.

Os modelos de grafo para os principais métodos (‘main’, ‘run’, ‘verificaDivisibilidade’, ‘verificaDigitosContidos’ e ‘imprimeProprioNumero’) contribuíram para uma representação visual clara do fluxo de execução, simplificando a compreensão da estrutura e interações do código.

Durante a realização deste projeto, aprendi a importância da abordagem de testes unitários na garantia da qualidade do código. Também foi possível aprofundar o conhecimento sobre a utilização de princípios ágeis, como a simplicidade e a clareza do código. A implementação modular e incremental do programa garantiu a facilidade de manutenção e a compreensão do código, tornando-o mais robusto e escalável.

Por fim, para concluir, ao abordar estes conceitos todos, fortaleci não só a confiança no desenvolvimento de código, mas também fiquei a compreender melhor sobre as boas práticas de desenvolvimento de testes unitários a serem aplicados em projetos futuros.