Pontifícia Universidade Católica de Minas Gerais
Bacharelado em Engenharia de Software
Teste de Software

Alunos:

Geovane de Freitas Queiroz Morcatti
Guilherme Henrique Ladislau Biagini
Lucas Araujo Borges de Lima
Marcos Henrique Dias Barbosa
Paulo Angelo Dias Barbosa
Weber Marques de Oliveira

Prof. Johnatan Alves de Oliveira

Belo Horizonte

2022

1.JUnit

1.1 O que é

JUnit é um framework orientado a objetos, desenvolvido para a linguagem Java, que serve como base para o desenvolvimento e execução de testes unitários. Ela define a API TestEngine para o desenvolvimento da estrutura de teste e aplicações gráficas em modo console para executar os testes criados.

1.2 Versões suportadas

JUnit 5 requer Java 8 (ou superior) em tempo de execução. No entanto, você ainda pode testar o código que foi compilado com versões anteriores do JDK **(Kit de Desenvolvimento Java**).

1.3 Notações

Os casos de teste são definidos por meio de notações que determinam a estrutura do teste. Algumas delas são:

`@Teste`: Esta notação indica que um método é um método de teste. Observe que esta anotação não possui nenhum atributo.

`@ParameterizedTest`: Os testes parametrizados tornam possível executar um teste várias vezes com argumentos diferentes. Eles são declarados apenas como regulares.

`@BeforeEach`: Esta notação denota que o método notado deve ser executado antes de cada método de teste, análogo ao JUnit 4's.

1.4 Métodos de teste

Métodos de teste são qualquer método de instância que é anotado diretamente ou meta-anotado com **`@Test`**, **`@RepeatedTest`**, **`@ParameterizedTest`**, **`@TestFactory`**, e **`@TestTemplate`**.

1.5 Ordem de execução

Por padrão, as classes e métodos de teste serão ordenados usando um algoritmo determinístico, mas intencionalmente não óbvio. Isso garante que execuções subsequentes de um conjunto de testes executem classes de teste e métodos de teste na mesma ordem, permitindo compilações repetíveis.

1.6 Ciclo de vida

Para permitir que métodos de teste individuais sejam executados isoladamente e evitar efeitos colaterais devido ao estado inesperado da instância de teste mutável, o JUnit cria uma nova instância de cada classe de teste antes de executar cada método de teste.Esse ciclo de vida da instância de teste "por método" é o comportamento padrão no JUnit Júpiter e é análogo a todas as versões anteriores do JUnit.

1.7 IDE Suportadas

IntelliJ IDEA, Eclipse, NetBeans, and Visual Studio Cod

2.Sistema

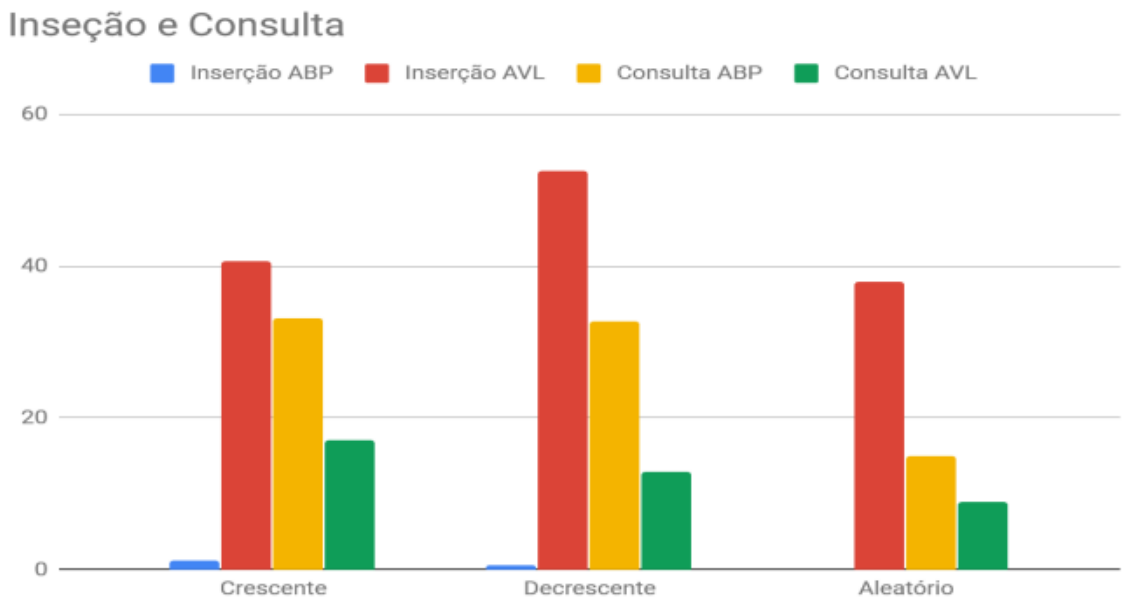
O sistema consiste em uma aplicação com algoritmos de ordenação que verifica qual o mais lento em relação ao tamanho dos vetores.

Relatório - Parte 1

Ao implementar e testar as operações de inserção e consulta em uma Árvore Binária de Pesquisa (ABP) e em uma Árvore AVL, foi medido o tempo de execução de cada um dos métodos. Os testes foram realizados na linguagem de programação Java em um desktop com 4096MB de RAM. Com os dados agrupados a fim de facilitar a visualização e análise, foi gerada uma tabela e um gráfico no Excel que compara o tempo obtido (milissegundo) em cada uma das 4 operações sobre 3 vetores, sendo eles:

Vetor A : preenchido com os valores entre 1 e 10.000.000(ou o maior tamanho possível);
Vetor B : preenchido com os valores entre 10.000.000 e 1 (vetor invertido);
Vetor C: preenchido aleatoriamente entre 1 e 10.000.000;

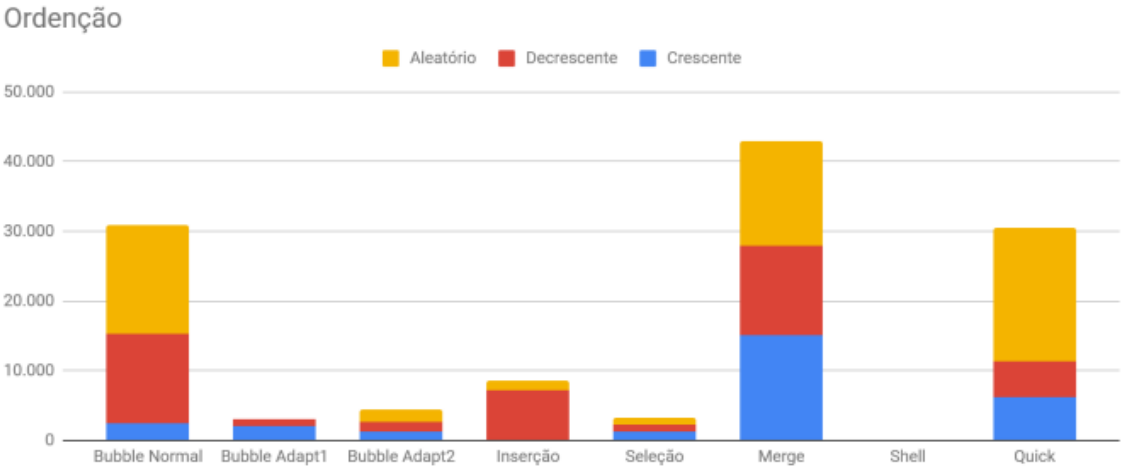
	Parte 1			
	Inserção ABP	**Inserção AVL**	**Consulta ABP**	**Consulta AVL**
Crescente	1,25	40,635	33,15	17
Decrescente	0,55	52,512	32,677	13
Aleatório	0.15	38	15	9



Comentários: Após programa rodar por aproximadamente 4 horas sem apresentar nenhum resultado diminuímos a quantidade de dados de 1000000 para 100000 dessa maneira conseguimos apresentar resultados rápidos e que tem um valor representativo.

Relatório - parte 2

Parte 2								
	Bubble Normal	**Bubble Adapt1**	**Bubble Adapt2**	**Inserção**	**Seleção**	**Merge**	**Shell**	**Quick**
Crescente	2.516	2.000	1.317	0.2	1.205	15.000	00.40	6.146
Decrescente	12.685	1.000	1.310	7.249	1.047	13.000	00.20	5.249
Aleatório	15.684	0	1.871	1.293	1.009	15.000	00.07	19.112



Comentários: Era esperado que o QuickSort fosse o mais rápido porém não foi bem assim que o resultado saiu, o ShellSort apresentou um melhor desempenho. Durante os testes o QuickSort deu erro de stack overflow e duas maneiras de resolver foram usadas, uma foi dar um shuffle nos dados e o outro foi aumentar a memória utilizada para evitar o erro.

3.Testes

EclEmma

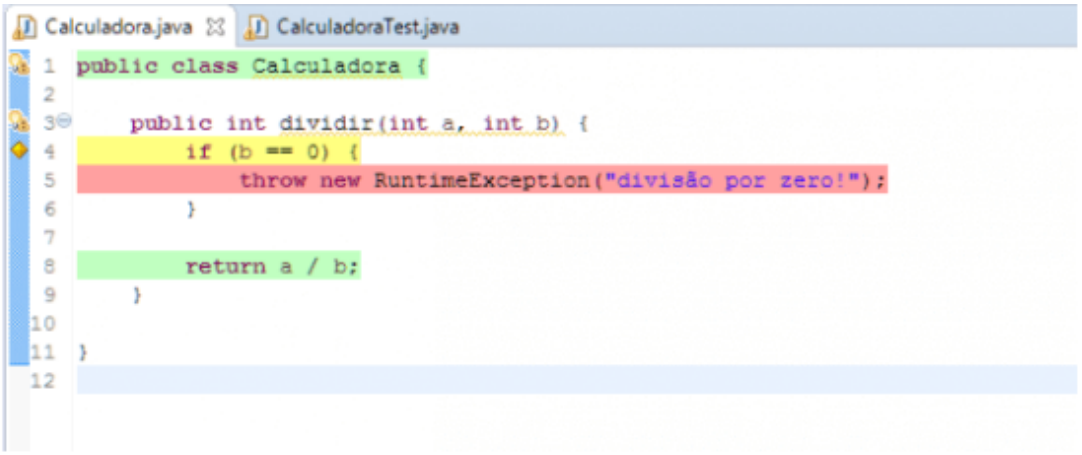
O EclEmma é uma ferramenta de cobertura de código para Java. De forma resumida, ele te mostra qual a porcentagem do seu código seus testes estão cobrindo. Com esta ferramenta fica claro identificar a parte do código que não está sendo coberto em nenhum dos seus testes unitários.

- **Ciclo de desenvolvimento/teste rápido:** Lançamentos de dentro do ambiente de trabalho, como execuções de teste JUnit, podem ser analisados diretamente para cobertura de código.
- **Análise de cobertura avançada:** Os resultados da cobertura são imediatamente resumidos e destacados nos editores de código-fonte Java.
- **Não invasivo:** EclEmma não requer a modificação de seus projetos ou qualquer outra configuração

O EclEmma é um plugin gratuito, disponível para o Eclipse sob a Eclipse Public License. A ferramenta foi inspirada na biblioteca EMMA desenvolvida por Vlad Roubtsov. A partir da versão 2.0, foi baseada na biblioteca JaCoCo de “cobertura de código”. O JaCoCo está disponível para qualquer desenvolvedor que tenha interesse em criar sua própria ferramenta e está disponível no site oficial. Até o momento em que escrevo deste artigo, a última versão disponível é a 2.3.3.

Dentre as principais características que posso citar da ferramenta, a principal delas é que o EclEmma funciona de forma independente do seu projeto e pode ser executado a qualquer momento. Além disso, ela também dá suporte a diversos frameworks de testes, tais como:

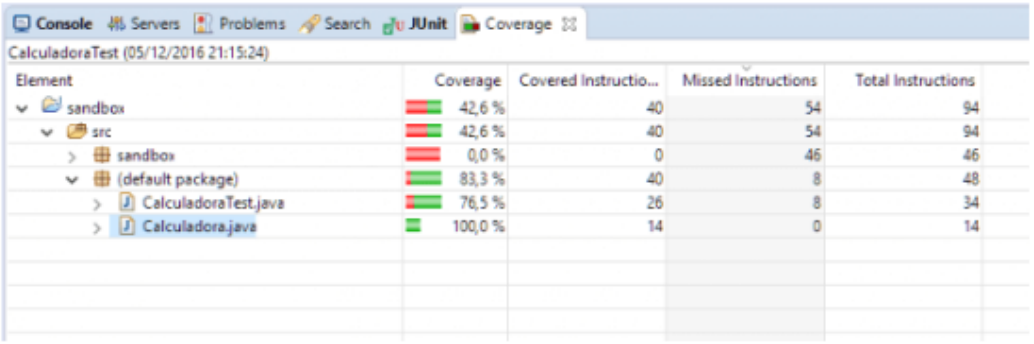
- JUNit
- TestNG
- SWTbot



Verde: Código executado

Amarelo: Ponto de decisão

Vermelho: Código não executado



Exemplo: Mostrando o coverage da Cobertura.

Métricas de Cobertura que ele reconhece:

https://www.eclemma.org/jacoco/trunk/doc/counters.html

4.Referências

https://drive.google.com/drive/folders/1XX38IDSlHI91Rz9gFHS8ZGkIGPf6hJc3?usp=sharing
https://www.devmedia.com.br/testes-de-integracao-com-java-e-junit/25662
https://jakarta.apache.org/cactus/
https://github.com/noconnor/JUnitPerf

5.Link do projeto

https://github.com/lucasABLima/TP2-Teste