JIC JORNADA DE INICIAÇÃO CIENTÍFICA E TECNOLÓGICA
SIMPÓS SUL III Simpósio de Pós-Graduação do Sul do Brasil
17 a 19 de outubro
UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

TESTE BASEADO EM PROPRIEDADES APLICADOS PARA FERRAMENTAS DA LINGUAGEM JAVA

JOÃO PEDRO LAMAISON[1], CASSIANE GAIARDO[2], SAMUEL FEITOSA[3]

1 Introdução

Durante um projeto de Iniciação Científica no IFSC, foi desenvolvido um gerador de código Java. Esse gerador inclui instruções de controle de fluxo e permite a geração de acesso a propriedades, invocação de métodos, criação de objetos, conversão de tipos, entre outras funcionalidades. Com essa ferramenta, é possível gerar construções básicas da linguagem Java e realizar testes com propriedades definidas (KRAUS, SCHAFASCHEK, RIBEIRO, FEITOSA, 2021).

O objetivo do projeto é facilitar o processo de criação de código Java e permitir a realização de experimentos com diferentes construções da linguagem. Com essa ferramenta, os pesquisadores e estudantes podem explorar várias características do Java e testar diferentes cenários.

É importante ressaltar que testes manuais não são sempre confiáveis, pois em um sistema de grande porte é possível que mesmo desenvolvedores esqueçam partes ou regras de sua aplicação e isso pode gerar problemas diversos. Portanto é essencial que existam testes automatizados desde o mais simples até pontos mais complexos do sistema, dessa forma podemos elevar a qualidade do produto e também o índice de confiabilidade do sistema (KRAUS, SCHAFASCHEK, FEITOSA, 2021).

Nesse projeto pretende-se desenvolver o tratamento de entradas de código com base no conceito de "Typed Holes Programming" (Programação com Buracos Tipados) que é um conceito utilizado em linguagens de programação funcionais e estáticas para auxiliar no desenvolvimento de código durante a fase de escrita e refatoração. Em linguagens funcionais estáticas, como Haskell, OCaml, Hazelnut e Elm, é comum encontrar a inferência de tipos, na qual o compilador deduz os tipos das expressões do código com base no contexto e nas

[1] Graduando em Ciência da Computação, Bolsista UFFS, UFFS, Câmpus Chapecó, joao_lamaison@hotmail.com
[2] Graduanda em Ciência da Computação, UFFS, Campus Chapecó
[3] Doutor em Ciência da Computação, UFFS, Campus Chapecó

restrições do programa (OMAR, VOYSEY, CHUNGH, HAMMER, 2019).

Um "hole" (buraco) é uma espécie de marcador ou espaço reservado que o programador pode deixar em seu código, nesse caso espaços marcados com “?” como por exemplo “?int?”, onde o tipo da expressão ou função é conhecido, mas a implementação ainda não foi escrita. O compilador reconhece esse buraco e, durante a compilação, irá sinalizar que ele está incompleto, indicando o tipo esperado no local.

Essa abordagem é especialmente útil para prototipar códigos, pois permite que o programador se concentre na estrutura geral do programa, deixando as partes específicas ou mais complexas para depois. Também ajuda na refatoração de código existente, permitindo que o desenvolvedor veja o tipo esperado em um determinado contexto e, assim, ajuste o código conforme necessário. Porém nesse caso estará fazendo o papel de gerar variáveis e testar métodos do mesmo tipo, que então serão passados para as funções da aplicação com o intuito de verificar se tudo está conforme deveria estar (YUAN, GUEST, GRIFFIS, POTTER, MOON, OMAR, 2023).

2 Objetivos

O objetivo do presente trabalho é a continuação da elaboração de um gerador de programas aleatórios em java, de modo com que possamos elevar a qualidade de uma aplicação por meio de testes automatizados. O desenvolvimento desse pedaço do projeto utiliza o conceito de Programação de Buracos Tipados, conceito mais presente em linguagens funcionais como Haskell, mas que é possível de ser feito em Java com a ajuda de meios como regex.

Objetivos específicos

- Realizar uma revisão da literatura sobre o tema abordado.
- Identificar as principais ferramentas de isolamento de código e substituição.
- Criar um meio de realizar esse isolamento e substituição.

3 Metodologia

O projeto começa com uma pesquisa exploratória do código do gerador (KRAUS, SCHAFASCHEK, RIBEIRO, FEITOSA, 2021) para delinear o escopo e a compreensão dos aspectos aplicados ao domínio do problema deste trabalho. Essa pesquisa permitiu entender como e onde devem ser feitas as alterações que serão apresentadas nos resultados parciais.

É importante salientar que os resultados obtidos devem auxiliar na continuação do

projeto e pode ser utilizado para testar diferentes partes do mesmo.

Esse projeto tem a seguinte proposta de metodologia:

Fase 1: Levantamento bibliográfico do projeto. Pesquisando e compreendendo diferentes conceitos sobre as partes do gerador, para então começar a encontrar os locais que deveriam ser alterados para encaixar o algoritmo de troca de código marcado.

Fase 2: Estudo sobre ferramentas, bibliotecas e outros métodos que poderiam ajudar e facilitar o desenvolvimento.

Fase 3: Começo do desenvolvimento marcando trechos de código e criando métodos que recebam e façam as alterações.

Fase 4: Compreensão dos resultados obtidos e busca de melhorias.

4 Discussão e Resultados Parciais

Em sua versão atual, já foi obtida ao menos uma maneira de demarcar trechos de código a serem substituídos e também uma maneira de obter os tipos e efetuar a substituição, agora resta estudar se esse método é o mais efetivo e, portanto, será necessário testar suas limitações e continuar o desenvolvimento do projeto sempre buscando novas e melhores formas de programar a solução.

```
String code = "public class Exemplo {\n" +
              "    int num = ?int?;\n" +
              "    String name = ?String?;\n" +
              "    double value = ?double?;\n" +
              "}";
```

Com essa entrada de exemplo, após a obtenção dos tipos basta substituirmos a parte demarcada por métodos ou variáveis que respeitem o mesmo tipo.

```
public static String replaceTypes(String code, List<String> types){
       for (String type:types){
       String value = getReplacementValue(type);
     code=code.replace("?"+type+ "?", value);
}return code}
```

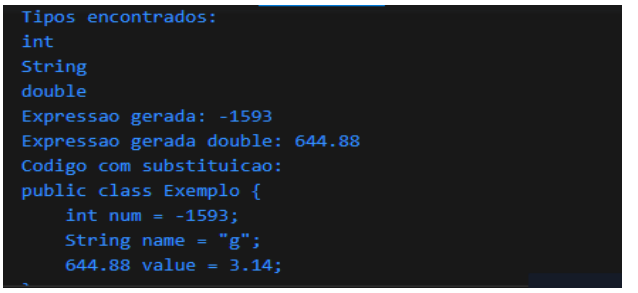
O replaceTypes representa o método no qual passaremos a String do código e a lista de tipos, e então para cada tipo obtido será chamado o método que irá gerar os valores dos tipos obtidos, e após retorno realizará a troca dos trechos marcados por meio da regex replace.

```
public String getReplacementValue(String type) { // verifica o tipo e
retorna,
        switch (type) {
            case "int":
                Arbitrary<Expression> e = mCore.genExpression(
                        mCtx,
                        ReflectParserTranslator.reflectToParserType("int"));
                String printar = e.sample().toString();
                System.out.println("Expressao gerada: " + printar);
                return printar.toString();

            case "String":
                Arbitrary<LiteralExpr> s = mBase.genPrimitiveString();

                return s.sample().toString();

            case "double":
                Arbitrary<Expression> doubles = mCore.genExpression(
                        mCtx,

ReflectParserTranslator.reflectToParserType("double"));
                String printarDouble = doubles.sample().toString();
                System.out.println("Expressao gerada double: " +
printarDouble);
                return printarDouble.toString();
            // Adicionar mais casos para outros tipos, se necessário
            default:
                return "null";
        }}
```

Por meio do método sample, iremos retirar um valor aleatório dentre os valores gerados (através dos métodos genExpression, genPrimitiveString) para o tipo em questão, de modo que iremos passar esse valor para a substituição no Replace Types.

Após o processamento do código em questão, é apresentado um relatório dos tipos encontrados, e um código é gerado automaticamente para preencher os espaços reservados.

Imagem 4- Retorno do método com os tipos encontrados e substituição realizada.



Fonte: Do autor

5 Conclusão

Finalmente, no estágio atual do projeto, é perceptível que ainda existe trabalho a ser feito, mas as bases do desenvolvimento foram lançadas, encontramos maneiras de demarcar, obter as marcações e também substituir essas marcações. É importante salientar a necessidade do estudo contínuo desses métodos para encontrar suas limitações e identificarmos se realmente se encaixam nos requisitos necessários para alcançarmos nosso objetivo principal.

Referências Bibliográficas

KRAUS, Luiz; SCHAFASCHEK, Bruno; FEITOSA Samuel. **Desenvolvimento de um Gerador de Programas Aleatórios em Java**, Santa Catarina, 2021 .

KRAUS, Luiz; SCHAFASCHEK, Bruno; and RIBEIRO, Rodrigo Geraldo and da Silva; FEITOSA, Samuel **Synthesis of Random Real-World Java Programs from Preexisting Libraries,** Santa Catarina, 2021

OMAR, Cyrus; VOYSEY, Ian; CHUNGH, Ravi; HAMMER, Matthew A; **Live functional programming with typed holes,** New York, NY, USA, 2019

YUAN, Yongwei; GUEST, Scott; GRIFFIS, Eric; POTTER, Hannah; MOON, David and OMAR, Cyrus, **Live Pattern Matching with Typed Holes,** New York USA, 2023

Palavras-chave: Geração de Programas Aleatórios; Teste Baseado em Propriedades; Programação com Buracos Tipados.

Nº de Registro no sistema Prisma: PES - 2022 - 0124

Financiamento

Este projeto foi desenvolvido com apoio financeiro para o pagamento de bolsas de pesquisa pela Universidade Federal da Fronteira Sul - UFFS, através do edital nº 89/GR/UFFS/2022.