



Trabalho 2 (T2) [Peso 4]

Data de entrega:
15/05/2019 (quarta-feira)

1. Modo de entrega e desenvolvimento

A entrega e o desenvolvimento do trabalho se darão exclusivamente com git via serviço [Github](#). Quando começar o trabalho, siga os seguintes passos:

1. [Crie um repositório privado](#) no Github com o nome do seu projeto;
2. [Adicione os usuários](#) do seu grupo nesse repositório, mais o usuário `Dovycki` (ou dovycki@gmail.com);
3. Cada integrante do grupo clona o repositório para a sua máquina (ou máquina do lab) e trabalha normalmente, fazendo commits e pushes regulares.
4. No arquivo `README.md` do projeto, coloque a) as informações exigidas para projeto e b) nome dos integrantes do grupo.



IMPORTANT! Não faça um commit único com todo o código do projeto. Faça um commit para cada passo do projeto, e.x. criação da estrutura inicial, adição de um novo elemento, etc. Não fique com medo de ter vários commits. **Ter um único commit com tudo pronto é um problema.** O histórico de commits mostrará a sua evolução no projeto. Você pode ter commits até as 23:59 horas da data de entrega do trabalho.

1.1 Estrutura do repositório e dos arquivos

O repositório deve ter, obrigatoriamente, a seguinte estrutura de pastas e arquivos (imagine que o nome da linguagem criada para esse interpretador chama-se "blah"):

- `fonte/`
- `exemplos/`
- `manual.md`
- `blah.jar`

O arquivo `manual.md` é um documento descrevendo a linguagem. Esse documento deve ser escrito em formato [markdown do Github](#). Ele deve ter os comandos da linguagem, sua sintaxe, exemplos, etc, de forma que um programador consiga escrever um programa utilizando **APENAS** o manual como guia. O arquivo `blah.jar` é o interpretador empacotado no formato JAR (vide Capítulo 11 da apostila). Se a sua linguagem se chamar *Abobrinha*, por exemplo, seu interpretador está empacotado como `abobrinha.jar`. A pasta `fonte` contém o código fonte do interpretador. A pasta `exemplos` contém programas de exemplo escritos na linguagem do interpretador.

Trabalhos que não seguirem esse formato receberão **nota zero**.

1.2 Revisão periódica não avaliativa do código-fonte (*code review*)

O professor fará uma revisão **não avaliativa** do código fonte de cada projeto em duas ocasiões. A revisão será conduzida através de comentários feitos no próprio código usando-se a interface web do Github (similar ao que está descrito [aqui](#)). O objetivo dessa revisão não é dar nota ao trabalho, mas sim apontar possíveis problemas de implementação, ajudar a cumprir as exigências do trabalho, prevenir impacto negativo na nota final, etc.

A primeira revisão acontecerá aproximadamente 14 dias após o início do projeto. A segunda revisão acontecerá aproximadamente 22 dias após o início do projeto.

2. Descrição

O trabalho pode ser desenvolvido em grupos de 3 ou 4 alunos (os requisitos do trabalho mudam conforme o número de participantes, veja a **Tabela 1**). Implementar um programa em Java que interprete uma linguagem de



Universidade Federal da Fronteira Sul
Ciência da Computação
Programação I - Prof. Fernando Bevilacqua

programação criada pelos alunos. O programa deve funcionar da seguinte forma pela linha de comando (veja seção 5. Dicas):

```
java NomePrograma arquivo.ext
```

sendo que o parâmetro `arquivo.ext` será o arquivo a ser interpretado pelo programa, que pode ter qualquer nome e qualquer extensão, a critério dos alunos. O interpretador deve ser capaz de analisar e reagir corretamente às seguintes situações no arquivo fonte que ele esteja interpretando:

- Declaração de variáveis;
- Atribuição de valor a variável;
- Expressões com no mínimo dois operandos, ex.: `a + b`;
- Operações de adição, subtração, divisão e multiplicação;
- Laço;
- Comando de saída, ex.: mostrar algo na tela;
- Controlador de fluxo, ex.: `if`;
- Aninhamento de comandos, ex.: `ifs` dentro de `ifs`, `laços` dentro de `laços`.

3. Avaliação

Os seguintes elementos serão considerados durante a correção:

- Funcionamento correto (o programa precisa cumprir seu objetivo conforme a descrição do trabalho);
- Legibilidade do código (nomes de classes com a primeira letra maiúscula, métodos e propriedades no formato `nomeFormaCamelo`, **indentação correta**, etc). Trabalhos mal indentados sofrerão desconto considerável em sua nota;
- Utilização de forma satisfatória dos conceitos de orientação a objetos vistos em aula;
- Comentários (o código fonte deve conter um bloco de comentário no começo informando o propósito do programa e o nome/email do seu autor).
- O interpretador deve ser testado, no mínimo, com dois programas: um que calcula a média de dois números e outro que diz se um número é primo ou não.
- Histórico do controle de versão é condizente com o trabalho realizado (Ex. vários commits espalhados ao longo do tempo, não um único commit com diversos arquivos).
- Mostrar domínio sobre a implementação durante o seminário de apresentação para a turma.

3.1 Observações

1. Haverá um desconto de 50% da nota do trabalho por dia de atraso na entrega, com prazo máximo de 3 dias de atraso;
2. Trabalhos copiados receberão nota zero e o nome dos envolvidos será levado ao colegiado do curso;
3. Programas que não compilarem receberão nota zero instantânea (nenhuma avaliação será realizada).



IMPORTANTE! Esse trabalho será utilizado como base para trabalhos futuros. Isso quer dizer que a sua nota em alguns dos trabalhos futuros impactada se você não entregar esse trabalho.

3.2 Implementação e pontuação

A pontuação no trabalho depende dos elementos da linguagem que for implementada, bem como do número de participantes do grupo. Isso quer dizer que um grupo com 3 ou 4 integrantes recebe notas diferentes para um mesmo elemento que for implementado, por exemplo.



Universidade Federal da Fronteira Sul
Ciência da Computação
Programação I - Prof. Fernando Bevilacqua

A **Tabela 1** mostra quais funcionalidades/critérios serão considerados para avaliação, bem como a nota que eles rendem quando implementados por um grupo de 3 ou 4 alunos.

Tabela 1 - Pontuação

Funcionalidade na linguagem criada / critério	Nota (grupo de 3 alunos)	Nota (grupo de 4 alunos)
Declaração de variáveis; Atribuição de valor a variável; Expressões com no mínimo dois operandos; Operações de adição, subtração, divisão e multiplicação; Laço; Comando de saída (ex.: mostrar algo na tela); Controlador de fluxo (ex.: if);	0 a 6	0 a 3
Linguagem com aninhamento de comandos (ex.: ifs dentro de ifs, laços dentro de laços).	6 a 8	3 a 5
Comandos de entrada	+ 0.5	5 a 6
Sintaxe flexível (ex. não exigir número determinado de espaços entre comandos)	+ 0.5	6 a 8
Funções (com escopo)	+10%	+5%
Vetores	+5%	0.5
Utilização e demonstração de entendimento dos conceitos de orientação a objetos.	8 a 10	8 a 10

4. Forma de apresentação

O trabalho deve ser apresentado, em aula, em um seminário realizado para a turma inteira. Cada grupo terá direito a 15 min de apresentação, na qual deverão, obrigatoriamente, cobrir os seguintes tópicos:

- Breve apresentação da linguagem (nome, sintaxe, funcionalidades, repositório);
- Como cada linha do código-fonte é analisada (ex. como os elementos são “quebrados” em partes, como o interpretador sabe que uma linha é uma atribuição, etc.);
- Como funciona, em termos de implementação, o sistema de variáveis;
- Como funciona, em termos de implementação, o sistema de expressões;
- Como funciona, em termos de implementação, ifs e laços.
- Demonstrar o interpretador funcionando com, no mínimo, 2 programas. Durante a demonstração, alterações serão exigidas (ex.: “crie uma laço naquela posição”, “adicione uma variável naquela expressão”, etc).

A nota da apresentação será individual. Cada membro do grupo será questionado sobre partes do trabalho, tendo que responder sozinho à pergunta.

5. Dicas

5.1 Testes do interpretador

O programa deve ser capaz de interpretar qualquer cenário possível dentro dos comandos que disponibiliza, ou seja, ele precisa interpretar qualquer código escrito com a linguagem criada, não apenas um programa em específico. Para armazenar as variáveis, talvez seja interessante criar uma classe `Variavel` com atributos `nome` e `valor`. O programa principal terá um vetor de objetos `Variavel`, sendo que cada entrada nesse vetor corresponde a uma variável do programa sendo interpretado.

Para garantir que seu interpretador funciona em diversos cenários, teste ele com os códigos abaixo (que foram escritos em C, você precisa converter para a sua linguagem):



Universidade Federal da Fronteira Sul
Ciência da Computação
Programação I - Prof. Fernando Bevilacqua

```
// Olá Mundo
#include <stdio.h>
void main() {
    printf("Ola mundo!");
}
```

```
// Declarações e atribuições
#include <stdio.h>
void main() {
    int a;
    a = 2;
    printf("%d", a);
}
```

```
// Laço
#include <stdio.h>
void main() {
    int i;
    i = 0;
    while(i < 10) {
        printf("%d", i);
        i++;
    }
}
```

```
// Expressões
#include <stdio.h>
void main() {
    int a, b;
    a = 1 + 1;
    b = 3 + a;

    printf("%d", a);
    printf("%d", b);
}
```

```
// Ifs
#include <stdio.h>

void main() {
    int a;
    a = 1;

    if(a > 1) {
        printf("Maior");
    } else {
        printf("Menor");
    }
}
```

```
// Aninhamento
#include <stdio.h>
void main() {
    int i;
    int j;
    i = 0;
    j = i;
    while(i < 3) {
        if(i == 0)
            printf("Zero");
        else
            printf("Outro");
        i++;
    }
    if(i == 3) {
        if(j == 0)
            printf("Ok!");
        else
            printf("Erro");
    }
}
```



Universidade Federal da Fronteira Sul
Ciência da Computação
Programação I - Prof. Fernando Bevilacqua

5.2 O parâmetro `args` da `main()`

O parâmetro `args` do método `main()` é um vetor de strings. Cada uma das entradas desse vetor é uma palavra passada pela linha de comando quando o programa é chamado. Por exemplo, dada a classe `Main` abaixo:

```
public class Main {  
    public static void main(String[] args) {  
        int i;  
  
        if (args.length == 0) {  
            System.out.println("Rode: java Main param1 param2");  
        }  
  
        for(i = 0; i < args.length; i++) {  
            System.out.println("args[" + i + "] = " + args[i]);  
        }  
    }  
}
```

Se você executar o seguinte comando:

```
java Main ola mundo
```

O vetor `args` terá na posição 0 a palavra "ola" e na posição 1 a palavra "mundo". Você também pode passar frases inteiras em cada uma das posições de `args`, basta usar aspas duplas na linha de comando. Ex.:

```
java Main "Ola mundo" "frase grande aqui"
```

Nesse caso, o vetor `args` terá na posição 0 a string "Ola mundo" e na posição 1 a string "frase grande aqui". Você precisará receber pela linha do comando (`args`) o caminho até o arquivo que o seu interpretador irá executar.



Universidade Federal da Fronteira Sul
Ciência da Computação
Programação I - Prof. Fernando Bevilacqua

5.3 Leitura de arquivo

A classe `Scanner` do Java pode ser usada para ler o conteúdo de um arquivo, conforme o exemplo abaixo:

```
import java.io.File;
import java.util.Scanner;

public class MeuScanner {
    public static void main(String[] args) {
        try {
            File file = new File("teste.txt");
            Scanner input = new Scanner(file);

            while (input.hasNextLine()) {
                String line = input.nextLine();
                System.out.println(line);
            }

            input.close();

        } catch (Exception e) {
            System.out.println("Nao foi possivel abrir o arquivo teste.txt.");
            System.out.println("Ele existe mesmo?");
            e.printStackTrace();
        }
    }
}
```