Linguagem para questionários interativos

Universidade de Aveiro

Beatriz Rodrigues 92023, Bruno Lopes 89179,
Daniel Correia 90480,
Diogo Correia 90327, Lara Rodrigues 93427

Linguagem para questionários interativos

DEPARTAMENTO IV

Universidade de Aveiro

Beatriz Rodrigues 92023, Bruno Lopes 89179, Daniel Correia 90480,
Diogo Correia 90327, Lara Rodrigues 93427

Junho de 2020

Resumo

A linguagem desenvolvida neste projeto tem como premissa a criação de um questionário interactivo que será posteriormente embutido numa linguagem de programação. Quando o programa é executado são apresentadas perguntas de escolha múltipla, este espera pela resposta dada pelo utilizador e guarda os resultados que serão exibidos num relatório de resultados no final da execução.

Agradecimentos

Queremos agradecer aos professores da unidade curricular de Linguagens Formais e Autómatos, André Zúquete e Artur Pereira pela ajuda na interpretação e concessão do projeto, e ao professor Miguel Oliveira e Silva pela elucidação teórica das ferramentas. Esta ajuda levou-nos a abordar o mesmo da melhor forma. Adquirimos conhecimentos sobre compiladores e interpretadores durante a realização deste trabalho, demonstrando assim que este projeto final é indispensável para um melhor aprimoramento dos conteúdos da cadeira de Linguagens Formais e Autómatos.

Conteúdo

Capítulo 1

Introdução

Este documento aborda a forma como foi criado e programado o interpretador e o compilador que origina a linguagem de programação final.

O relatório está dividido em duas partes fundamentais,o Capítulo 2 é dedicado à descrição do interpretador e serve também como documentação de apoio do projeto para facilitar a utilização das linguagens.

O Capítulo 3 tem como principal objetivo descrever como é possível utilizar a linguagem criada.

Capítulo 2

Linguagens

2.1 Quiz

2.1.1 Blocos de Ações - Main e Funções

Para uma estruturação mais clara da linguagem, foi feita uma abordagem de separação da mesma em blocos de ações. Existe o bloco principal, a Main, que controla todos os outros blocos secundários, as funções. A lingugem tem obrigatoriamente de ter uma Main para correr, mas não precisa de ter Funções.

Para inicializar a **Main** é usada a keyword main, seguida da partícula => para abrir o bloco e da partícula » para fechar. Dentro do bloco são colocadas as várias ações.

Exemplo prático(1):

```
main =>
        text title => "Quiz of questions";
        call makeMenu(title, 3);
>>
```

Para inicializar uma **Função**, para além do uso da keyword function, é também preciso especificar os parâmetros de entrada, se algum, e o nome da mesma.

Exemplo prático(2):

```
function (text quizTitle, text nmrOptions) makeMenu =>
        write (CONSOLE) => quizTitle;
        ...
>>
```

Para que uma Função tenha efeito, esta tem de ser chamada na Main através da keyword call seguida do nome e dos valores que vão servir os argumentos,

como observador em **Exemplo prático(1)**, através da linha de código ***call makeMenu(title, 3);***

Uma Função pode ainda devolver um valor, de qualquer tipo. Esse valor pode depois ser acedido onde quer que a função tenha sido chamada. Para o fazer é só usar a keyword return seguida do que quer que se queira devolver.

Exemplo prático(3):

```
function (text quizTitle, text nmrOptions) makeMenu =>
        ...
        return title;
>>
```

Para aceder ao valor devolvido por uma Função, é só atribuir a chamada dessa função a uma variável, como demonstrado na linha de código ***text title => call makeMenu(title, 3);***

2.1.2 Variáveis

- **text -** é usada quando se quer guardar informação formada por um conjunto de caracteres, texto, equivalente ao tipo String em Java.

  Exemplo prático(1):

  ```
  text title => "Quiz of questions";
  ```

  Existe também a possibilidade de facilmente concatenar várias variáveis deste tipo, ou até mesmo literais, semelhante ao que é feito em Java.

  Exemplo prático(2):

  ```
  text title => "Quiz"+"of"+"questions";
  text option => "["+count+"]"+optionText;
  ```

- **number -** no caso da necessidade de armazenar valores numéricos, é usado number, que equivale ao tipo de dados int em Java.

  Exemplo prático:

  ```
  number n => 12;
  ```

- **boolean -** permite armazenar uma das keywords TRUE ou FALSE, que representam respetivamente o valor lógico verdadeiro (1) e falso (0). Equivalem ao tipo com o mesmo nome em Java.

Para negar um valor lógico é usada a keyword NOT, equivalmente ao ! em Java.

Exemplo prático:

```
boolean hasQuestion => TRUE;
boolean hasTitle => FALSE;
if (hasQuestion) => --considerando o valor da variável definido anteriormente, isto vai correr >>
if (NOT hasTitle) => --considerando o valor da variável definido anteriormente, isto vai correr >>
```

- **question -** uma question é um tipo nativo de dados que representa uma Classe específica em Java. A question possui um conjunto de atributos acessíveis através dos seguintes comandos:
  - var.**title** - permite o acesso ao título
  - var.**right** - permite o acesso às respostas certas
  - var.**wrong** - permite o acesso às respostas erradas
  - var.**difficulty** - permite o acesso à dificuldade das respostas (EASY, MEDIUM, HARD)
  - var.**type** - permite o acesso ao tipo de questão (MULTIPLE, OPEN)
  - var.**tries** - permite o acesso ao número de tentativas permitidas para responder a cada questão
  - var.**points** - permite o acesso ao número de pontos de cada questão

2.1.3 Listas

Quando implementadas, as listas criam arrays list de qualquer um dos tipos existentes na linguagem, ou seja, text, number, boolean ou question e são definidas como apresentado em seguida.

Exemplo prático:

```
main =>
        list boolean li => {FALSE, TRUE, FALSE};
        list number li2 => {1, 12, 3};
        list text li3 => {"efwwe"};
        list text li4;
        li4 => {"sad", "asd"};
        list question li5;
>>
```

2.1.4 Outputs

Existem duas opções para que sejam criados outputs, uma delas é o que em Java corresponde ao ***"System.out.print"*** que imprime no terminal uma qualquer mensagem ou variável e cujo exemplo é apresentado a seguir. Para além desta, existe a opção de escrever o conteúdo desejado num ficheiro. O código para implementar esta função é similar ao da função de print, a única diferença é que em vez do token CONSOLE o utilizador terá de inserir o ***path("../example.txt")*** do ficheiro.

- **Prints**

  Exemplo prático:

```
main =>
        write (CONSOLE) => "Faz print";
>>
```

- **Write in files**

  Exemplo prático:

```
main =>
        write (path) => "Escreve esta linha no ficheiro ";
>>
```

2.1.5 Inputs

Também existe a possibilidade de fazer a leitura de argumentos vindos do teclado ou da leitura de ficheiros. Estas duas ações têm, novamente, implementações semelhantes tal como nos outputs e os seus exemplos são apresentados em seguida.

- **Read from keyboard**

```
main =>
        text n => read (CONSOLE);
        questions => import (n);
>>
```

- **Read from files**

```
main =>
        text n => read (path);
        questions => import (n);
>>
```

2.1.6 Ciclos

Os ciclos desenvolvidos nesta linguagens são bastante semelhantes aos já existentes em Java, desde a sua implementação até à forma como se comportam.

- **forIn-**Neste ciclo a condição é definida dentro dos parênteses e é definida na forma for(var in var) sendo que a condição será a variável estar contido em algo também definido pelo utilizador, como por exemplo um array.

  Exemplo prático:

```
number n => 7;
    for(number n in arr) =>
        n++;
        write (CONSOLE) => n;
    >>
```

- **forTo-** Este ciclo for é semelhante ao anterior, aqui também a condição é apresentada dentro de parênteses, mas em vez da variável estar contida em algo, a condição será percorrer um intervalo definido pelo programador, podendo este ser crescente ou decrescente (<, >, <=, >=).

```
number n => 7;
    for a (1 < n) =>
        a++;
        write (CONSOLE) => a;
    >>
```

- **doaslong-** Este ciclo pode ser equiparada a um ciclo ***"do while"*** existente em Java.O ciclo será feito enquanto a condição definida dentro dos parênteses do *aslong* for verdade.

  Exemplo prático:

```
main =>
    text a => "TRUE";
    number n => 0;
    do =>
        n++;
    >>aslong("a"!= "a");
>>
```

- **aslong-** Tal como no ciclo anterior, também este foi desenvolvido de forma semelhante ao ciclo ***"while"*** do Java.

  Exemplo prático:

```
main =>
        text a => "TRUE";
        aslong (a) =>
                write (CONSOLE) => "Infinite aslong loop";
        >>
>>
```

2.1.7 Condicionais

As atribuições condicionais podem ser usadas pelos ciclos e pelas implementações condicionais, como por exemplo, ***o if,o elsif e o final***. Estas condições poderão ser : **'AND','OR','==','!=','>' ou '<'**.

Exemplo prático:

```
main =>
        number n => 0;
        number a => 2;
        boolean c => FALSE;
        boolean d => FALSE;
        if("a"!= "sdfds") =>
                n++;
        >>
        if("a"== "a") =>
                text b => "a";
        >>
        final =>
                write (CONSOLE) => "a";
                n+=5;
        >>

        if(n != a) =>
                n++;
        >>
        elseif(n > a) =>
                n- -;
        >>
        elseif(c and d) =>
                n/=2;
```

```
        >>
        final =>
                write (CONSOLE) => "a";
                n+=5;
        >>
>>
```

2.1.8 Aritmética

Esta linguagem dispões de várias operações aritméticas que poderão ser usadas em qualquer situação de forma a trabalhar com as variáveis. Estas operações poderão ser atribuições de sinal, somas, divisões, multiplicações e atribuição de prioridades através do uso dos parênteses.

Exemplo prático:

```
main =>
        number a=> 12;
        number b => a*4;
        number res => b/a;
        write (CONSOLE) => res;
        number res -=12;
        write (CONSOLE) => res;
        callparens(b);
>>

function(number n1) parens =>
        number n => ( 2 + 2 ) * 2;
        write(CONSOLE) => n;
>>
```

2.2 Questions

2.2.1 Perguntas e atributos

O propósito da linguagem Questions é criar uma estrutura de dados para armazenar as perguntas que o programador queira colocar no seu questionário, bem como as suas características. Os dados são armazenados num Hashmap, onde a uma key(ID) corresponde um objecto(pergunta). Uma pergunta é composta pelos seguintes atributos:

- ID - frase/palavra/numero delimitado por : '<' e '>' destinado a identificar a pergunta posteriormente.
- Titulo - correspode à pergunta em si, é o que é apresentado ao utilizador que realiza o questionário.
- Tipo - uma pergunta pode ser de resposta **ABERTA** (tem apenas um conjunto de respostas corretas), de escolha **MÚLTIPLA** (tem respota(s) correta(s) e respostas erradas).
- dificuldade - uma pergunta tem 3 níveis de dificuldade: **EASY**, **MEDIUM** e **HARD**.
- tentativas - numero de tentativas que o utilizador tem para responder corretamente à pergunta. É necessário definir um valor default, caso não seja especificado da pergunta.
- Pontos - pontuação atribuída a cada pergunta. É necessário definir um valor default, caso não seja especificado da pergunta.
- Respostas corretas - conjunto de respostas corretas (possivelmente apenas uma resposta)
- Respostas erradas - conjunto de respostas erradas (pode ser um conjunto vazio, não necessáriamente declarado).

Exemplo prático de definição de perguntas:

```
<Capitalpt> { title: "Qual a capital de portugal?",
        difficulty: EASY,
        tries: 3,
        points: 15,
        right: [Lisboa],
        wrong: [Madrid , Roma , Paris] }
```

Exemplo prático de definição de defaults:

```
        tries: 3
        points: 10
```

2.2.2 Classe Question e HASHMAP

A classe Question tem atributos estáticos que armazenam informação geral das questions, nomeadamente as variáveis de default (defaultTime, defaultTries e defaultPoints) assim como o HASHMAP(Estrutura de armazenamento de dados que vai ser usado no Quiz) que é criado em runtime e guarda uma Key(id),Value(Question) sempre que um novo objeto do tipo Question é criado.

2.2.3 Gramática e interpretador

Tendo como objetivo fornecer um conjunto de perguntas (criado à priori) à linguagem principal do Quiz, esta linguagem(Questions) é mais simples e limitada.O programador apenas pode desenvolver perguntas ou definir os valores de default. Ambos tem uma ordem estruturada que tem que ser seguida, embora algumas características da pergunta possam ser omitidas ou não declaradas.

Tendo apenas 3 tokens não terminais, o interpretador faz primeiramente a definicão dos defaults através da função setDefault da Classe Question, seguido pela construção das perguntas(faz a "tradução"de tokens terminais para variáveis do tipo String ou int, além de utilizar os setters da classe para guardar as caracteristicas da pergunta).

Capítulo 3

Especificações da linguagem

3.1 Palavras Reservadas

As palavras reservadas da linguagem do Quiz são:

- text
- number
- question
- boolean
- list
- map
- main
- function
- clone
- split
- get
- put
- delete
- import
- clear
- read
- write

- writeln
- remove
- add
- random
- for
- do
- aslong
- if
- elsif
- final
- in
- call

Capítulo 4

Instruções de Utilização

Dependendo da posição do utilizador perante este projeto, poderão ser necessárias diferentes abordagens. Se a intenção do utilizador é criar um questionário, terá de trabalhar num ficheiro .qst para fazer as questões e num ficheiro .quiz para fazer o questionário. É importante que os ficheiros estejam sempre nos devidos diretórios, isto é, os ficheiros .qst no diretório ./questions, os ficheiros .quiz no diretório ./quiz e garantir que o Compilador e o Interpretador, ambos se encontram no diretório ./source.

Para correr um ficheiro .quiz, deve efetuar os seguintes comandos no Terminal:

```
$ java QuizMain nomeficheiro.quiz NomeClassOutputJava
$ javac NomeClassOutputJava.java
$ java NomeClassOutputJava
```

Após este conjunto de comandos, o programa deverá começar.

Capítulo 5

Observações

Apesar da linguagem do Quiz (Quiz) ser bastante completa e dinâmica existiam ainda algumas particularidades da linguagem que gostaríamos de ter desenvolvido para que o programa fosse ainda mais interessante. Desenvolvemos algumas funções que seriam interessantes ter no questionário, mas por não estarem 100% funcionais no nosso Quiz decidimos deixar comentadas.

Relativamente à linguagem destinada para as perguntas (Questions) , o maior "problema"é o facto de esta ser limitada. As perguntas devem ser feitas de acordo com uma estrutura que, dando alguma liberdade ao programador, também é algo limitadora. O programador tem que seguir uma ordem específica na declaração dos atributos da pergunta, por exemplo. Em ambas as linguagens uma das maiores dificuldades foi "lutar"contra a ambiguidade nas gramáticas.

Capítulo 6

Contribuições dos autores

A contribuição dos membros do grupo para a realização do projeto foi semelhante sendo de 20% para cada um. O trabalho foi divido por todos e houve bastante entreajuda entre nós.