Estudo sobre a Linguagem de Programação Java

Vancarla Soares e Marivaldo Samuel

[1]Departamento de Ciência da Computação – Universidade Federal de Roraima (UFRR)
Boa Vista – RR – Brasil

vancarlasoaresaquino@gmail.com, mmarivaldo27@gmail.com

Resumo. Este relatório tem como objetivo descrever as características da Linguagem de Programação Java, abordando seu histórico, suas instruções e funcionalidades.

História do surgimento da linguagem

Java é uma linguagem de programação interpretada orientada a objetos desenvolvida na década de 90 por uma equipe de programadores chefiada por James Gosling, na empresa Sun Microsystems. Diferente das linguagens de programação convencionais, que são compiladas para código nativo, a linguagem Java é compilada para um *bytecode* que é interpretado por uma máquina virtual (Java Virtual Machine, mais conhecida pela sua abreviação JVM). A linguagem de programação Java é a linguagem convencional da Plataforma Java, mas não é a sua única linguagem.

A história começa em 1991, em San Hill Road empresa filiada a **Sun** (da qual hoje pertence a empresa Oracle), formado pelo time de engenheiros liderados por **Patrick Naugthon**, **Sun Fellow** e **James Gosling**.

O grupo estava iniciando um projeto denominado **Projeto Green**, que consistia na criação de tecnologias modernas de software para empresas eletrônicas de consumo. A ideia principal do Java era que os aparelhos eletrônicos se comunicassem entre si. Por exemplo, o caso de possuir um fogão, você poderia deixar assando sua comida e quando estivesse pronta iria enviar uma mensagem para o micro-ondas ligar e após isso tocar o seu despertador, sendo algo do gênero.

Com o tempo perceberam que não poderiam ficar presos aos sistemas operacionais, até porque os clientes não estavam interessados no tipo de processador que estavam utilizando, e sim na tecnologia. Portanto para o grupo criar uma versão do projeto para cada tipo de sistema era inviável, sendo assim, foi desenvolvido o sistema operacional **GreenOS**.

A linguagem de programação chamada de **Oak** (carvalho) foi criada pelo chefe do projeto **James Gosling**. A explicação da origem do nome foi que enquanto pensava numa estrutura de diretórios para a linguagem, observava pela janela um carvalho. Mas esse nome já estava registrado, então o nome acabou surgindo na cafeteria local da cidade onde tomavam café. “Java”, pois era o nome da terra de origem do café, que os programadores da equipe apreciavam nessa cafeteria, por isso que a logo do Java é um café.

Em 1993, apareceu uma oportunidade para o grupo Green. A empresa FirstPerson junto com a Time-Warner estava pedindo propostas de sistemas operacionais de decodificadores e tecnologias de vídeo sob demanda. Foi na época em que o NCSA apresentou o MOSAIC 1.0, o primeiro navegador gráfico para Web. Então a empresa FirstPerson apostou nos testes da TV da Time-Warner, mas esta empresa acabou escolhendo a tecnologia oferecida pela Silicon Graphics.

Em 1995 a Sun viu uma oportunidade na Web, nessa época nas páginas não existia muita interatividade, apenas conteúdos estáticos eram exibidos. Então nesse ano a Sun anunciou o ambiente Java, sendo um absoluto sucesso, gerando uma aceitação aos browsers populares como o Netscape Navigator e padrões tridimensionais como o VRML (Virtual Reality Modeling Language - Linguagem de Modelagem para a Realidade Virtual).

O Java foi o primeiro a utilizar decodificares de televisões interagindo em dispositivos portáteis e outros produtos eletrônicos de consumo, foi do mesmo jeito que foi iniciado em 1991, possuindo portabilidade para qualquer ambiente e do desenvolvimento para múltiplas plataformas, em ambientes de eletrônicos de consumo, desde então o Java vem liderando o mercado em termos de linguagem.

2. Domínios de aplicação

Linguagem de Programação JAVA, está presente em diversos aplicações como nos dispositivos móveis com androide, web (sites) e a computação nas nuvens (servidores). Ex. Celulares, PDA's, TVs, Sensores.

3. Paradigmas suportados pela linguagem

Imperativo: Descreve a computação como ações, enunciados ou comandos que mudam o estado (variáveis) de um programa. Muito parecido com o comportamento imperativo

das linguagens naturais que expressam ordens, programas imperativos são uma sequência de comandos para o computador executar.

Orientado ao objeto: Programação orientada a objetos (também conhecida pela sua sigla POO) é um modelo de análise, projeto e programação de sistemas de software baseado na composição e interação entre diversas unidades de software chamadas de objetos. É uma maneira de programar que ajuda na organização e resolve muitos problemas enfrentados pela programação procedural.

Exemplo:

```
public class Cachorro{

        public String nome;

        public int peso;

        public String corOlhos;

        public int quantPatas;

}
```

Funcional: trata a computação como uma avaliação de funções matemáticas e que evita estados ou dados mutáveis. Ela enfatiza a aplicação de funções, em contraste da programação imperativa, que enfatiza mudanças no estado do programa.

Exemplo:

```
public class IteracaoExemplo1 {

    public static void main(String[] args) {

        List<String> valores = Arrays.asList("1", "2", "3");

        Iterator<String> it = valores.iterator();

        int soma = 0;

        while(it.hasNext()) {

            soma += Integer.parseInt(it.next());
```

```
        }

        System.out.println("Soma = " + soma);

    }

}
```

Genérica: Os algoritmos são escritos em uma gramática estendida de forma a adaptar-se através da especificação das partes variáveis que são definidas na instância do algoritmo. Especificamente, a gramática estendida eleva um elemento não variável ou uma construção implícita na gramática base para uma variável ou constante, permitindo a utilização do código genérico.

Estruturada: Forma de programação de computadores que preconiza que todos os programas possíveis podem ser reduzidos a apenas três estruturas: sequência, decisão ou seleção e iteração (esta última também é chamada de repetição), desenvolvida por Michael A. Jackson no livro "Principles of Program Design" de 1975.

Reflexiva: A programação reflexiva, também chamada de programação orientada à reflexão, é usada para escrever programas no paradigma reflexivo. Este, é usado como uma extensão para o paradigma da orientação a objeto, para adicionar auto otimização e aumentar a flexibilidade de um aplicativo. Nesse paradigma a computação não é trabalhada somente durante a compilação do programa, mas também durante sua execução. Outras abordagens imperativas, tais como os paradigmas da programação procedural ou orientada a objeto, especificam que há uma sequência pré-estabelecida de operações (sejam elas funções ou chamadas de métodos), que modificam qualquer dado a elas submetido.

Concorrente: Construção de programas de computador que fazem uso da execução concorrente (simultânea) de várias tarefas computacionais interativas, que podem ser implementadas como programas separados ou como um conjunto de threads criadas por um único programa. Essas tarefas podem ser executadas por um único processador, vários processadores em um único equipamento ou processadores distribuídos por uma rede. Programação concorrente é relacionada com programação paralela, mas foca mais na interação entre as tarefas. A interação e a comunicação correta entre as diferentes tarefas, além da coordenação do acesso concorrente aos recursos computacionais são as principais questões discutidas durante o desenvolvimento de sistemas concorrentes.

Exemplo:

```
public class ExemploThread {

  // Mostra uma mensagem de acordo com o nome da thread atual   static void mensagemDaThread(String messagem) {

    String nomeDaThread =

    Thread.currentThread().getName();

    System.out.format("%s: %s%n",nomeDaThread,messagem);

  }



  private static class LoopDaMensagem implements Runnable {

    public void run() {

      String mensagens[] = {

       "Lpoo é a primeira matéria de java",          "Depois vem ALPOO",          "Finalmente quem sobrevive",          "Ai sim pode pegar o diploma"

      };



      try {

        for (int i = 0; i < mensagens.length; i++) {

        // Pause for 4 seconds

          Thread.sleep(4000);

         // Print a message

          mensagemDaThread(mensagens[i]);

        }

      } catch (InterruptedException e) {

        mensagemDaThread("Thread não finalizada!");       }

    }
```

```
  }


  public static void main(String args[])throws InterruptedException {

    mensagemDaThread("Iniciando a thread com o loop da mensagem");          long startTime =
System.currentTimeMillis();

    Thread[] t = new Thread[5];

    for(int i=0;i<4;i++){

      t[i]= new Thread(new LoopDaMensagem());

      t[i].start();

    }


    MensagemDaThread("Esperando a thread terminar");     mensagemDaThread("Finalmente!");

  }

}
```

4. Variáveis e tipos de dados

Dados Numéricos

Tipos Inteiros:

Tipo	Memória consumida	Valor Mínimo	Valor Máximo
byte	1 byte	128	127
short	2 bytes	32.768	32.767
int	4 bytes	2.147.483.648	2.147.483.647
long	8 bytes	9.223.372.036.854.775.808	9.223.372.036.854.775.807

Dados Textuais

É possível representar dois tipos de elementos textuais em Java: caracteres e textos. A representação de um caractere solitário é feita pelo tipo char e a representação de textos é feita pela classe String.

Enquanto o tipo de char representa apenas um caractere, a representação de textos deverá ser feita pela classe String. Essa classe pode ser utilizada de forma similar aos tipos primitivos, mais os valores literais desse tipo são transcritos entre aspas e não entre apóstrofos.

Dados Lógicos

O tipo lógico é representado, em Java, pelo tipo booleano. Este tipo pode armazenar um de dois valores possíveis: true ou false. Ele é empregado para realizar testes lógicos em conjunto com operadores relacionais e dentro de estruturas de decisão e repetição. O tipo boolean do Java equivale ao tipo boolean do Pascal.

Variáveis

Uma variável representa a unidade básica de armazenamento temporário de dados e compõe-se de um tipo, um identificador e um escopo. Seu objetivo é armazenar um dado de determinado tipo primitivo para que possa ser recuperado e aplicado em operações posteriores.

Para compreender como funciona o uso de variáveis é preciso analisar como elas são criadas, de que modo recebem e armazenam dados e como estes são recuperados. Também é importante entender onde elas podem ser declaradas e onde podem ser utilizadas, tendo em vista seu escopo.

Constantes

As constantes são unidades básicas de armazenamento de dados que não devem sofrer alterações ao longo da execução do aplicativo. O uso de constantes é menos frequente que o uso de variáveis. No entanto, há situações em que elas são requeridas e, por isso, é indispensável entender o que são, para que servem e como podem ser utilizadas.

A declaração de uma constante contém apenas um elemento a mais que a declaração de uma variável: a palavra reservada final. Assim como as variáveis, as constantes

compõem-se de um tipo, um identificador e um escopo. Veja a sintaxe que determina como declarar e inicializar uma constante:

final < tipo > < identificador > = < valor >;

Mas enquanto a declaração e a inicialização de variáveis podem ser feitas em instruções distintas, toda constante deve ser declarada e inicializada em uma única instrução e, depois disso, não lhe pode ser atribuído outro valor.

Além de tudo que foi dito sobre constantes, é importante observar uma convenção no momento de declará-las: devem ser utilizadas somente letras maiúsculas para seu identificador. A aderência a essa convenção torna o código-fonte muito mais legível, facilitando a distinção entre variáveis e constantes.

4. Comandos de controle

Estruturas de controle de decisão

Estruturas de controle de decisão são instruções em linguagem Java que permitem que blocos específicos de código sejam escolhidos para serem executados, redirecionando determinadas partes do fluxo do programa.

If - Else

O comando if permite que valores booleanos sejam testados. Se o valor passado como parâmetro para o comando if for true(verdadeiro) o bloco do if é executado caso contrário o bloco do else é executado. O parâmetro passado para o comando if deve ser um valor booleano, caso contrário o código não compila. O comando else e o seu bloco são opcionais.

Exemplo:

```
public class Controles {

  public static void main(String[] args) {

    int nota1 = 4;

    int nota2 = 3;

    int resultado = nota1 + nota2;

    if (resultado > 6) {
```

```
      System.out.print("Aluno aprovado");
    } else {
      System.out.print("Aluno reprovado");
    }  //se resultado maior(>) que 6, aluno aprovado,
   //senao, aluno reprovado
  }
}
```

While

Através do comando while, é possível definir quantas vezes um determinado trecho de código deve ser executado pelo computador.

Exemplo:

```
public class Controles {
  public static void main(String[] args) {
     int x = 0;     // O do-while garante que pelo  menos uma vez o codigo seja executado
     do {
      System.out.println(x);
      x++;
    } while (x < 10);   } }
```

For

A declaração for, como nas declarações anteriores, permite a execução do mesmo código uma quantidade determinada de vezes.

For possui três argumentos:

```
for (declaração_inicial; expressão_lógica; salto) {
instrução1;
```

instrução2; ...

}

onde:

declaração_inicial: inicializa uma variável para o laço

expressão_lógica: compara a variável do laço com um valor limite

salto: atualiza a variável do laço.

Exemplo:

/* Neste exemplo, uma variável i, do tipo int, é inicializada com o valor zero. A expressão lógica

"i é menor que 10" é avaliada. Se for verdadeira, então a instrução dentro do laço é

executada. Após isso, a expressão i terá seu valor adicionado em 1 e, novamente, a condição

lógica será avaliada. Este processo continuará até que a condição lógica tenha o valor falso. */

```
public class Controles {
  public static void main(String[] args) {
     for (int i = 0; i < 10; i++) {
      System.out.print("item: " + i);
    }
 }
```

Comandos de controle de fluxo

Break

O comando break é um comando bastante importante no desenvolvimento da maior parte dos programas de computador, ele é usado para sair imediatamente de um laço (loop, em inglês), independentemente do valor de CONDIÇÃO. Ele pode ser executado dentro de um while, for, do ... while ou switch fazendo uma saída imediata dessa instrução. Passando para a execução do próximo comando.

A sintaxe do comando é bastante simples: **break;**

Exemplos de uso: No exemplo abaixo temos um código escrito em Java, onde em um loop for é interrompido quando a variável inteira contador se torna igual a 5.

```
 public class BreakTeste

{

   public static void main( String args[] )

   {

   int contador; //Variável de controle usada como referência



   for ( contador = 1; contador <= 10; contador++ )//Laço, será repetido 10 vezes

      {

      if ( contador == 5 ) //Se o contador chegar até 5

         break;        //Termina o loop quando a condição do if se tornar verdadeira



      System.out.printf( "%d ", contador);

      }//Termino da instrução for



   System.out.printf( "\nInterrompe o contador quando o contador = %d\n",contador );

   }//Fim do main

}//Fim da classe BreakTest
```

Continue

O comando continue serve para encerrar a execução de comandos e verificar o valor de CONDIÇÃO. Caso o valor seja "verdadeiro", a iteração continua. Caso contrário, ela se encerra. Exemplos de uso:

```
 for(int i=1;i<=10;i++){        //O loop é executado 10 vezes
```

```
    if(i%2==0)
        continue;
    System.out.println(i+" "); //Será impressa na tela os números ímpares entre 1 e 10
}
```

5. Escopo (regras de visibilidade)

Especificadores de controle de acesso

Há quatro graus de visibilidade que podemos usar com membros de uma classe As palavras chaves usadas são: "public", "private", "protected", e nenhuma Para entender quando usar cada um desses graus de visibilidade, lembre-se que há vários papeis que os programadores podem assumir:

- Você, que está escrevendo uma classe.
- O programador "cliente" que só quer usar a classe que você criou.
- Ele pode nem ter código fonte.
- Outros programadores que estão trabalhando num pacote com você.
- Várias classes de um mesmo pacote (packages) estão sendo feitas por vários programadores.
- O programador que poderá estender sua classe no futuro.
- Ele normalmente tem o código fonte.

A visibilidade public

- Quem tem acesso à classe tem acesso também a qualquer membro com visibilidade public.
- O alvo aqui é o programador cliente que usa suas classes.
- É raro ter atributos públicos, mas é comum ter métodos públicos.

A visibilidade private

- O membro private não é acessível fora da classe.
- A intenção aqui é permitir que apenas você que escreve a classe possa usar esse membro.

A visibilidade protected

- O membro protected é acessível à classe e a suas subclasses.
- A intenção é dar acesso ao programadores que estenderão sua classe.

Packages

- O conceito de package (pacote) foi inventado para permitir criar um espaço de nomes grande em Java
- O que ocorre se você quiser criar uma classe Fita e outro programador já criou uma classe Fita?
- Será que você vai ser impossibilitado de criar sua classe?
- Não há problema, desde que as classes assim chamadas estejam em packages diferentes
- Os nomes dos packages formam uma árvore, permitindo assim um espaço de nomes muito grande.
- Exemplo: p1.aplic.banco é um nome de package
- Exemplo: p1.aplic.banco.Conta é uma classe deste package

A visibilidade "package"

- Um membro de classe sem especificador de controle de acesso é dito ter a visibilidade package (ou "friendly")
- É como public, mas somente dentro do package Todas as classes do package podem acessar um membro "friendly"
- É usado para permitir acesso mais liberal, mas somente dentro de um mundo controlado e não pelos usuários da classe
- Deve-se ter cuidado com a visibilidade friendly para atributos pois pode abrir muito o acesso, principalmente em packages grandes

Um exemplo

- Observe a visibilidade de Agencia.fecharConta()

```
public class Agencia {

  ...

  /**

   * Fecha uma conta.

   * @param número O número da conta a fechar.

   * @throws NaoPodeFecharContaException se a conta não existir ou tiver saldo

   */

  /* observe visibilidade "package": tem que fechar a partir de fechar() da conta */

  static void fecharConta(int número) throws NaoPodeFecharContaException {

    abrirCaixa();

    Conta c = localizarConta(número);

    if(c == null) {

      throw new NaoPodeFecharContaException(c, "Conta nao existe");

    }

    if(c.getSaldo() != 0.0) {

      throw new NaoPodeFecharContaException(c, "Saldo nao esta zerado");

    }

    contas.remove(Integer.toString(número));

    // tem que repensar o fechamento de contas porque

    // do jeito que esta, nao some do arquivo agencia.txt

  }

  ...

}
```

Dentro do package p1.aplic.banco, várias classes poderiam chamar Agencia.fecharConta()
Porém, essa não é um método que queremos deixar público
Para fechar uma conta, o usuário do package deve usar Conta.fechar()

6. Exemplo prático de uso da linguagem de programação

O paradigma escolhido foi Orientação a Objetos. Esse paradigma foi escolhido por ter uma conexão direta com tudo que temos ao redor, e por seu ambiente de desenvolvimento ser mais interativo com a interface.

O ambiente de desenvolvimento

IDE Netbeans com JDKversão 32bytes, foi escolhido por ser uma IDE oficial que a ORA usa, e está diretamente ligado com o Java. A sua interface é mais simples e fácil de se trabalhar.

Descrição

Após ter instalado o Netbeans whit JDK, o programa é aberto e em arquivo é escolhido novo projeto. E automaticamente ele cria sistemas de arquivo. O netbeans possui uma área de desenvolvimento, com seus menus, e descrição do pacote, bibliotecas e testes.

7. Conclusão

Através deste relatório que por sua vez tem como objetivo retratar conceitos e aplicações de Java, foi possível obter conhecimento da linguagem de programação e ser feita a aplicação dela no projeto agenda telefônica, utilizando alguns de seus métodos abordados.

8. Referências

TABLELESS. **JAVA – Tipos de dados.** Disponível em: <https://tableless.com.br/java-tipos-de-dados/>. Acesso em: 20 jul. 2017.

Martins, Oberdan. **Estruturas de controle de decisão.** Disponível em: <http://aprendendojavajuntos.blogspot.com.br/2012/02/vetores-e-matrizesarrays.html> . Acesso em: 20 jul. 2017

WIKILIVROS. **Java/Comandos de iteração.** Disponível em: <https://pt.wikibooks.org/wiki/Java/Comandos_de_itera%C3%A7%C3%A3o> . Acesso em: 20 jul. 2017

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE. **Orientação a Objetos - Visibilidade.** Disponível em: <http://www.dsc.ufcg.edu.br/~jacques/cursos/p2/html/oo/visibilidade.htm>. Acesso em: 20 jul. 2017.

DE MEDIA. **Programação Funcional: código limpo e padrões de projeto.** Disponível em: <http://www.devmedia.com.br/programacao-funcional-codigo-limpo-e-padroes-de-projeto/32902>. Acesso em: 20 jul. 2017.