Pontifícia Universidade Católica de Minas Gerais
Linguagens de Programação

Java - Everywhere

Aluno(s): Eduardo Pereira Costa
Gustavo Gomes de Souza
Jonathan Douglas Diego Tavares
Stephanie Silva Vieira Gomes
Professor(a): Marco Rodrigo Costa

Maio
2021

Pontifícia Universidade Católica de Minas Gerais
Ciência da Computação
Linguagens de Programação

Java - Everywhere

Relatório de apresentação de conteúdo apresentado a disciplina de Linguagens de Programação, durante o Trabalho Teórico Prático, no Curso de Ciência da Computação da PUC-MG, como requisito final para entrega e avaliação do trabalho.

Alunos: Eduardo Pereira Costa
Gustavo Gomes de Souza
Jonathan Douglas Diego Tavares
Stephanie Silva Vieira Gomes

Professor: Marco Rodrigo Costa

Maio
2021

Conteúdo

1 Introdução

As linguagens de programação tornaram-se importantes elementos que compõem o universo da computação, sendo capazes de criar e alterar diversos aspectos da tecnologia. Nesse sentido, no presente documento será apresentada a linguagem Java, criada na década de 90 e atualmente uma peça chave no desenvolvimento de vários produtos tecnológicos.

Para conhecer as origens do Java, será abordado o histórico da linguagem, mostrando os criadores do projeto, as mudanças, evoluções, versões, utilizações e motivações para seu desenvolvimento.

A popularidade do Java cresceu ao longo dos anos, conquistando milhares de usuários e desenvolvedores. Ademais, os paradigmas da linguagem são expostos e discutidos, mostrando a classificação e características de cada um. Estão presentes em Java mais de dois paradigmas, sendo um deles, o Orientado a Objetos, que determina importantes aspectos da linguagem, e o funcional, que está sendo implementado nas versões mais atuais no intuito de introduzir ferramentas mais avançadas dentro da linguagem. Desta forma, é possível identificar as características mais marcantes presentes no Java, sendo cada peculiaridade destrinchada para que seja possível visualizar a sua relevância na linguagem.

Por fim, tem-se um comparativo com outras linguagens de programação como, por exemplo, C, Kotlin e Python. As diferenças e semelhanças foram exploradas e, através do contraponto entre as linguagens, foram analisadas vantagens, desvantagens, contextos e cenários em que Java pode ser mais interessante ou não em relação às demais.

Ao final, foi adicionado ao anexo exemplos de programas que permitem verificar aspectos da linguagem e algumas semelhanças que foram citadas no paralelo realizado com algumas linguagens similares/-confrontantes. O objetivo dos exemplos de programas foi demonstrar a flexibilidade e aplicabilidade da linguagem em diferentes cenários, reforçando o que fora dito nos tópicos abordados.

2 Histórico

Entre 1991 e 1994, a Sun Microsystems começou a trabalhar em uma nova linguagem de programação inicialmente denominada Oak como parte do Green Project. Os mentores do projeto eram Patrick Naughton, Mike Sheridan, e James Gosling. Para mostrar a tecnologia Oak e fazer com que seu software pudesse ser utilizado pelo máximo de pessoas possível, o time do Green Project desenvolveu seu próprio browser usando a linguagem Oak, o qual rodava aplicações feitas em Oak.

Os fundadores do Java iniciaram o projeto com a ideia de possibilitar diferentes aparelhos executarem um mesmo programa, independentemente da plataforma. Essa visão contribuiu para a noção de IoT (Internet of Things) que está fortemente presente na contemporaneidade.

Em 1995, na preparação para a sua primeira publicação, ou seja, acesso público a nova linguagem, a Sun Microsystems tentou registrar o nome Oak, mas este já havia sido registrado. Logo, em sessões de brainstorm, alguns nomes foram sugeridos, sendo a maioria descartados, até que houvesse apenas um, o Java. Em pouco tempo a nova linguagem ganhou sua logo, muito conhecida, a xícara de café.

Figura 1: Logo Java

O browser criado para rodar as aplicações Oak, agora Java[7], o WebRunner, foi demonstrado pela primeira vez em uma TED conference[25] (conferência para trocar ideias de Tecnologia, Entretenimento e Design) e foi renomeado para HotJava.

A primeira versão do Java é anunciada na conferência SunWorld, onde a Netscape também anunciou um browser com suporte para Java. A Netscape também licenciou o Java e lançou a primeira versão do browser Netscape Navigator com suporte a Java.

A Sun apresenta o slogan "Write Once, Run Anywhere" para descrever os recursos multiplataforma do Java. Em 1996, o JDK 1.0 (Java Development Kit) é lançado e o sistema operacional JavaOS é apresentado. O JavaOS é escrito principalmente em Java e se destina a sistemas incorporados. Ainda neste ano, a Sun apresenta seu primeiro programa focado em desenvolvedores, o Java Developer Connection.

Entre 1997 e 2000, o JDK 1.1 foi lançado junto com recursos principais como classes internas, JavaBeans, RMI Compiler, Reflection, o compilador Just-In-Time (JIT), internacionalização e suporte à Unicode. Logo após, o JDK 1.2 foi renomeado para Java 2 Platform, assumindo o nome de Standard Edition (J2SE 1.2). Novos recursos incluindo AWT, Java 2D, Swing e Coleções de Frameworks foram adicionadas a essa versão.

O J2SE 1.3 é lançado com mais uma leva de novos recursos incluindo o HotSpot Java VM, Java Naming, Interface de Diretório e a Java Platform Debugger Architecture. Mais de 100 milhões de smart cards com suporte a Java, foram vendidos ao final de 2000. O primeiro protótipo de Blu-ray CD Player com suporte para Java foi revelado.

2.1 2001 a 2005

Em 2001, o CEO da Oracle, Larry Ellison, juntou-se à liderança da Sun no palco do JavaOne para demonstrar um software capaz de executar Java, e ainda neste ano o J2SE 1.4 foi lançado com os recursos:

- Java Web Start;
- E/S sem bloqueio;
- API de registro;
- API de preferências;
- Expressões regulares.

Entre 2001 e 2004, no JavaOne, a equipe de desenvolvimento Java da Sun apresentou o primeiro prêmio anual Duke's Choice, que reconhece os inovadores que usam a tecnologia Java de uma maneira única. Ainda durante esse período, o site "java.com"foi lançado para permitir que os consumidores baixassem o Java Runtime Package, tornando possível executar aplicações Java em *desktops* e *laptops*.

Durante essa época, a NASA demonstrou um protótipo do Mars Rover no palco do JavaOne, mostrando como a tecnologia Java ajudaria a guiá-lo remotamente a partir do Centro de Controle de Missão da NASA. O J2SE 5.0 foi lançado em 2004 incluindo os recursos:

- Tipos Genéricos;
- Anotações;
- Enumerações;
- Variáveis como argumentos(varargs).

Além disso, ainda em 2004, o debate sobre o lançamento do Java como software de código aberto começou no palco do JavaOne e também nessa época, algum tempo depois da demonstração do protótipo, o Mars Rover Spirit da NASA pousou em Marte em 4 de janeiro de 2004, levando aplicativos Java para um novo planeta.

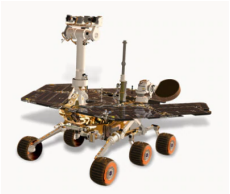

Figura 2: Mars Rover Spirit

Em 2005, a Sun estima que Java gera mais de US $100 bilhões em receita anualmente. Existem mais de 4,5 milhões de desenvolvedores Java, 2,5 bilhões de dispositivos que utilizam Java e 1 bilhão de smart cards baseados em Java. O programa Java Champions é lançado no JavaOne para reconhecer os líderes da comunidade de desenvolvedores.

2.2 2006 a 2017

Em 2006, a Sun lança o Java para o desenvolvimento de código aberto sob a "GNU General Public License", a mesma licença que rege o uso e o desenvolvimento do sistema operacional Linux, além disso o Java SE 6 também foi lançado neste ano. Ainda em 2006, o Java HotSpot VM e o compilador são lançados sob a licensa GNU. A imagem do mascote Duke torna-se de código aberto sob a "licença BSD".

Figura 3: Mascote do Java: Duke

Em setembro de 2008, o primeiro Java Virtual Machine (JVM) Language Summit foi realizado em Santa Clara, Califórnia.

Em 2009, a Oracle comprou a Sun e, sendo uma empresa que atuava basicamente na área de Dados, acabou ingressando também no universo do desenvolvimento de software e linguagens de programação. Larry Ellison, co-fundador e diretor executivo da Oracle, juntou-se à liderança da Sun no palco do JavaOne para falar sobre o compromisso da Oracle em investir na tecnologia Java para o benefício dos clientes e da comunidade. Entre 2009 e 2011, a Oracle lançou a Java Magazine, um blog de publicações técnicas sobre Java, escritas por desenvolvedores para desenvolvedores, e sendo um blog oficial da linguagem, fornece detalhes de novas aplicações Java, artigos técnicos de tutoriais, notícias da comunidade e muito mais. Em 2011, o Java SE 7 é lançado com novos recursos que incluem:

- Project Coin, que tem como objetivo, determinar qual conjunto de pequenas alterações na linguagem deve ser adicionado ao JDK 7;
- Invokedynamic, que permite que um compilador gere códigos com uma especificação mais flexível;
- Estrutura fork / join, que foi projetada para acelerar a execução de tarefas que podem ser divididas em outras subtarefas menores, executando-as em paralelo e combinando seus resultados para obter uma única resposta;
- Uma nova API de sistema de arquivos (NIO.2).

Em 2012, 97% dos computadores empresariais rodavam Java. Em 2014, o Java SE 8 foi lançado com recursos de expressões lambda, anotações de tipos em Java e uma API capaz de lidar com datas e horários. Ao final deste ano, mais de 9 milhões desenvolvedores ao redor do mundo usavam Java.

Em 2015, Java se tornou a plataforma mais utilizada para desenvolvimento no mundo, com mais de 10 milhões de desenvolvedores e 13 bilhões de dispositivos executando Java. O Twitter migrou sua infraestrutura para a JVM e passou a oferecer suporte a mais de 400 milhões de tweets por dia, aumentando o desempenho do aplicativo. No mesmo ano, a Netflix atendia a 2 bilhões de solicitações de conteúdo por dia com arquitetura baseada em Java.

Em 2016, 15 bilhões de dispositivos executavam Java enquanto plataforma principal, a Java Magazine tinha mais de 250.000 assinantes. A Uber começou a utilizar o Java como uma das principais linguagens para o desenvolvimento de sua plataforma com objetivo de tentar garantir alta performance.

Em 2017 o Java SE 9 foi lançado com novos recursos do Projeto Jigsaw (Java Platform Module System), sendo os principais:

- O Jshell, uma ferramenta de loop de leitura e avaliação;
- Compilação antecipada (Ahead-of-time Compilation), uma forma de melhorar o desempenho de programas Java e, em particular, o tempo de inicialização da JVM;
- Jlink, uma ferramenta que gera apenas os módulos da plataforma necessários para um determinado aplicativo;

- Strings compactas, uma forma de minimizar o gasto de memória na representação do que seria uma String na linguagem.

No mesmo ano, Java é ranqueada como a linguagem de programação número um no mundo, sendo a mais utilizada. Além disso, 12 milhões de desenvolvedores ao redor do mundo usavam Java, havendo 38 bilhões de JVMs ativas e 21 bilhões de JVMs conectadas à nuvem. No intuito de acelerar a inovação, a Oracle deu um prazo de seis meses para o lançamento do Java SE 10, que foi lançado em março de 2018.

2.3 2018 a 2020

Em 2018 Java foi ranqueada como a linguagem número um usada por desenvolvedores para a nuvem, neste mesmo ano o programa Java Champions alcançou 150 membros e o Java Community Process comemorou seu 20º aniversário. Ainda em 2018, a Oracle anunciou que abriria o código fonte de vários recursos Java comerciais, incluindo o compartilhamento de dados de classes de aplicativos, o Z Garbage Collector (ZGC), o Oracle Java Flight Recorder e o Oracle Java Mission Control.

O Java SE 10 foi lançado em 2018 após um período de 6 meses sem nenhum lançamento, incluindo recursos novos como:

- Inferência de tipo de variável local, que é um recurso que permite que o desenvolvedor ignore a declaração de tipo associada às variáveis locais, e o tipo é inferido pela JDK;
- Compartilhamento de dados de classe de aplicativo, um recurso que ajuda a reduzir o tempo de inicialização e o consumo de memória além do que já está sendo utilizado;
- Compartilhamento de dados de classe incluído no Java SE 5;
- Versionamento de lançamentos com base na data;
- O G1 Garbage Collector, utilizando execução totalmente paralela;
- Root Certificates;
- Thread-local Handshakes, que torna possível interromper threads individuais, além disso os thread-local handshakes reduzem a latência da JVM e melhoram sua eficiência.

Ainda em 2018, a Oracle aprimora as opções de licenciamento comercial do Java para desenvolvedores e empresas que desejam suporte de nível empresarial com a introdução do programa Java SE Subscription. O Java SE 11 também foi lançado em 2018 e incluía novos recursos como:

- HTTP client;
- Oracle Java Flight Recorder, uma ferramenta para coletar dados de diagnóstico e criação de perfil sobre uma aplicação Java em execução;
- Programas com código fonte de arquivo único;
- Transport Layer Security (TLS) 1.3, um recurso que criptografa os dados enviados pela Internet para garantir que hackers não interceptem o que está sendo transmitido.

Em 2019, Java foi mais uma vez classificado como a linguagem número um utilizada por desenvolvedores para a nuvem. No mesmo ano, o Java SE 12 foi lançado com novos recursos, que foram:

- Expressões Switch (first preview), que, como todas as expressões, avaliam um único valor e pode ser usado em declarações;
- JVM Constants API;

- Arquivos de compartilhamento de dados de classe padrão (CDS).

Ainda em 2019, o Java SE 13 é lançado com os recursos:

- Arquivos CDS dinâmicos;
- Capacidade de desalocar memória não utilizada;
- Switch Expressions (second preview);
- Blocos de texto (first preview).

Também em 2019, o Spotify escolheu o Java SE Subscription que elimina as preocupações da diretoria corporativa em relação à missão crítica, garantindo maior desempenho de software, estabilidade e atualizações de segurança.

Em 2020, o Java celebra seu 25º aniversário e o Java SE 14 foi lançado com mais alguns novos recursos, incluindo:

- Correspondência de padrões para instanceof (preview);
- Buffers de bytes mapeados não voláteis;
- NullPointerExceptions úteis, mais detalhadas;
- Records (preview);
- Switch Expressions (finalizadas);
- Blocos de texto (second preview);
- ZGC para macOS e Windows;
- API de acesso à memória externa;
- Ferramenta de pacotes.

Figura 4: Aniversário do Java

Ainda neste ano, o Java SE 15 foi lançado com mais alguns recursos novos, que incluíam:

- Classes seladas, que é uma classe ou interface que restringe quais outras classes ou interfaces podem estendê-la;
- Hidden Classes, um recurso que melhora a produtividade ao otimizar a forma como o Java funciona com frameworks que geram classes em tempo de execução e as usam indiretamente, via reflexão;
- Text Blocks, um recurso em versão prévia no JDK 13 e no JDK 14 que melhora a produtividade do desenvolvedor ao adicionar literais de cadeia de caracteres de várias linhas e formatar automaticamente cadeias de caracteres de uma forma previsível;

- Pattern matching para instanceof, um recurso em versão prévia, introduzido pela primeira vez no JDK 14 com o objetivo de melhorar a produtividade do desenvolvedor ao eliminar a necessidade de um código redudante, e deve permitir um código seguro mais conciso;

- Records, um recurso em versão prévia, introduzido pela primeira vez no JDK 14 com o objetivo de melhorar a produtividade do desenvolvedor ao fornecer uma sintaxe compacta para declarar classes que contêm dados superficialmente imutáveis.

Além disso, ainda em 2020 a Oracle lançou o "Inside.Java"oferecendo notícias e atualizações da Equipe Java da Oracle, e também lançou a série de podcasts "Inside.Java". Continuando, o Minecraft lançou um site onde havia cursos para ensinar pessoas com mais de 8 anos a programar em Java e criar seus próprios *mods* para o jogo utilizando a linguagem.

3 Paradigma

Cada linguagem de programação possui um conjunto de conceitos, princípios e regras e cada conjunto pode ser considerado como um paradigma. Paradigmas de linguagens de programação determinam conjuntos de modelos que possuem uma ou mais características em comum. Essas características são, por exemplo, o estilo ou estrutura de uma linguagem.

A linguagem Java pertence a mais de um paradigma, sendo eles o paradigma imperativo, orientada a objetos (POO), declarativo e funcional. Existem duas categorias principais de paradigmas:

- O imperativo, tendo duas subcategorias: POO e procedural.

- O declarativo, com duas subcategorias: funcional e lógico.

3.1 Paradigma Imperativo

O paradigma imperativo tem como características principais a execução sequencial de instruções e mudança do estado de um programa através de comandos. As instruções são uma definição de passos e determinam como o programa irá operar. As modificações de um estado são armazenadas enquanto variáveis na memória, e então, os comandos são executados sobre os dados armazenados (variáveis). Esse paradigma se preocupa em como alcançar o objetivo proposto e as instruções ditam como o programa opera para realizar esse propósito.

3.2 Paradigma Orientado a Objetos

No intuito de definir o que é o paradigma orientado a objetos, primeiro é necessário entender o que é um objeto. Foi dito que a definição de um objeto deriva de "uma entidade que contém dados e comportamentos"(WEISFELD, 2009, p.6). Entende-se um objeto como uma cápsula que tem operações que manipulam os dados pertencentes a ela. Em Java, essas operações são chamadas de métodos e os dados são chamados de atributos.

É necessário definir também o conceito de classe, que pode ser entendida como um modelo para um conjunto de objetos. Segundo WEISFELD[28], uma classe pode ser considerada como um template ou *blueprint* para um objeto, e desse modo, os objetos pertencem a esse template. As classes definem qual dado irá compor o objeto, sendo necessário especificar seus tipos e sua visibilidade. Já os métodos que podem ser executados sobre os dados dos objetos devem possuir uma assinatura, um conjunto de parâmetros e também sua implementação.

O principal objetivo do paradigma orientado a objetos é permitir o desenvolvimento de programas que possam representar objetos, entidades e sistemas reais, que sejam flexíveis, de fácil manutenção e que permitam reuso de "estruturas". Existem 4 princípios fundamentais que são base para esse tipo de programação, sendo eles: abstração, encapsulamento, herança e polimorfismo.

- *Abstração:* Abstração é o processo de identificar as qualidades ou propriedades importantes do fenômeno que está sendo modelado[22]. Um exemplo disso pode ser visto em um carro, em que para dirigir o carro não é necessário saber como os sistemas auxiliares funcionam, mas somente é preciso deter o conhecimento sobre os controles básicos (acelerar, frear, embreagem).

- *Encapsulamento:* Significa uma amarração entre os atributos (campos e métodos) de um objeto com uma ação[3]. A ideia é manter as propriedades e comportamentos de um objeto em um único local para que a manutenção e a extensão do código sejam possíveis e mais fáceis de serem realizadas. O encapsulamento também proporciona um mecanismo para que os atributos não possam ser acessados diretamente por uma fonte externa que foi declarada fora da classe de origem e abstrai detalhes que não precisam ser revelados aos usuários de objetos derivados da classe.

- *Herança:* É um mecanismo que permite a definição de novos objetos ou classes baseado no reuso de classes previamente existentes. É preciso existir uma classe base (classe pai) que define o comportamento geral para uma entidade. Cada subclasse, que deriva da classe base, podem herdar da mesma alguns comportamentos e atributos que já foram declarados, além de poder alterar/adicionar métodos já existentes ou mesmo criar novos métodos ou novos atributos.

- *Polimorfismo:* Em termos gerais, polimorfismo fornece uma opção ao uso de uma mesma interface sobre entidades de diferentes tipos[23].

3.3 Paradigma Funcional

O paradigma declarativo especifica o que um programa tem que fazer, sem definir como isso será alcançado. O paradigma funcional é um sub-paradigma do declarativo, nele as funções são similares as presentes na matemática, sendo a saída de cada função dependente apenas de seus argumentos, e sendo assim, o estado atual do programa não altera os argumentos e a forma com que os mesmos serão avaliados. Java tem suporte para esse paradigma em apenas alguns casos e por isso não é frequentemente associado ao paradigma funcional e declarativo, ou seja, não é uma linguagem funcional pura. Um exemplo de suporte a esse paradigma pode ser encontrado a partir da versão oito do Java, como o Streams.

```
1 IntStream.range(0, 10).filter(i -> i % 2 == 0).
2 forEach(System.out::println);
```

Listing 1: Uso de Streams

Os streams são definidos no pacote "java.util.stream"e são usados para gerenciar fluxos de objetos em que operações de caráter funcional podem ser feitas.

Para fazer esse mesmo exemplo de uma maneira imperativa, é possível utilizar o seguinte trecho de código usando Collections:

```
1  //Preenche um Arraylist com n elementos
2  List<Integer> lista = new ArrayList<Integer>();
3  for(int i = 0; i < n; i++){
4    lista.add(i);
5  }
6
7  //Preenche uma lista de pares atrav s um foreach sobre
8  //uma Collection que cont m valores inteiros
9  List<Integer> pares = new ArrayList<Integer>();
10 for(int val : lista){
11   if(val % 2 == 0) pares.add(val);
12 }
13
14 //imprime os elementos presentes na Collection pares
15 for(int val: pares){
```

```
16      System.out.println(val);
17  }
```

Listing 2: Uso de Collections

Percebe-se que foi necessário uma quantidade maior de linhas de código para alcançar o mesmo objetivo através do paradigma imperativo. O paradigma funcional tem a vantagem de ser menos "verboso", o que pode levar a um entendimento mais fácil do que está acontecendo durante a execução do programa.

Também foi introduzido no Java 8 o pacote "java.util.function", que segundo a documentação da linguagem,

> "As interfaces funcionais fornecem tipos para expressões lambda e referências de método. Cada interface funcional possui um único método abstrato, denominado método funcional para aquela interface funcional, ao qual o parâmetro da expressão lambda e os tipos de retorno são correspondidos ou adaptados. As interfaces funcionais podem fornecer um tipo em vários contextos, como contexto de atribuição, invocação de método ou contexto de cast".[15]

Exemplo:

```
1
2       // contexto de atribuicao
3       Predicate<String> p = String::isEmpty;
4
5       // contexto de invocacao de metodo
6       stream.filter(e -> e.getSize() > 10) ...
7
8       // contexto de cast
9       stream.map((ToIntFunction) e -> e.getSize()) ...
```

Listing 3: Exemplos de programação funcional em Java

4 Características mais marcantes

4.1 Java OO

Java surgiu como uma linguagem que está intimamente conectada ao paradigma orientado a objetos, sendo também incluída ao paradigma imperativo. Uma vez que Java lida com OO, ela torna-se uma linguagem capaz de explorar uma vasta extensão de mecanismos inerentes a esse paradigma. Todavia, Java não é uma linguagem puramente orientada a objetos, pois implementa também tipos primitivos de dados, os quais dentro da linguagem, não são representados enquanto objetos. Outra razão pela qual Java poderia não ser considerada puramente OO, é o fato de que atualmente, a linguagem implementa estruturas características do paradigma funcional, permitindo obter um desempenho muito superior quando se trata de contextos relacionados ao processamento paralelo de dados e recursividade, além de outros cenários pertinentes.

Pelo fato de Java ser uma linguagem que aplica fortemente os conceitos de OO, é possível utilizar-se da mesma para criar inúmeras *frameworks* que se aproveitam dos pilares básicos de OO implementados na linguagem. São exemplos de *frameworks* muito interessantes: "Spring", "Hibernate", "JUnit", "Ant", "Struts(J2EE)".

4.2 Portabilidade - JVM

Java é uma linguagem que foge aos padrões quando o assunto é o processo de compilação. É bastante conhecida pelo uso de uma máquina virtual (JVM), que por sua vez interpreta e executa programas através da geração de *bytecodes*, utilizando-se de um processador virtual e uma estrategia de compilação Just-In-Time que melhora o desempenho diminuindo o tempo necessário para a compilação.

Tal característica peculiar garante à linguagem a capacidade de executar qualquer programa independentemente da plataforma que estiver presente. Uma vez que a execução é feita pela JVM, torna-se possível alcançar uma portabilidade até então não antes vista em outras linguagens anteriores ao Java.

Ser portável garante também a possibilidade de executar software em dispositivos com hardware distintos, e sendo assim, é fácil compreender a razão pela qual o sistema operacional Android, desenvolvido pela Google e que roda Java para Android nativo, está presente em cerca de 71,8% dos dispositivos móveis por todo o mundo. Vale citar também que Java foi a linguagem utilizada para criar o jogo "Minecraft", que foi comprado pela Microsoft junto de sua produtora, a Mojang, pelo valor de aproximadamente $2,5 bilhões de dólares.

4.3 Flexibilidade da linguagem

Java é uma linguagem cuja aplicabilidade é extensa, estando presente desde o cenário empresarial até o entretenimento. São exemplos de possíveis cenários em que Java está sendo utilizado atualmente e fora utilizado no passado:

- Banco Inter, no cenário empresarial, em que Java é utilizado amplamente no backend dos sistemas que dão suporte a plataforma;
- Entretenimento voltado a jogos eletrônicos, tendo nomes como The Sims, Runescape, GTA: San Andreas(Mobile), Flappy Bird, Tibia e muito mais;
- Educação e aprendizagem de programação, sendo utilizado dentro de escolas e universidades para ensinar conceitos de lógica de programação e paradigma OO. Vale citar nesse contexto, que podem ser trabalhados os usos através do modo console ou interface gráfica;
- Na WEB, em que Java no passado foi amplamente utilizado para criar os famosos "Applets", que eram aplicações nativas em Java executadas pelos navegadores. Atualmente, é utilizado ainda mais na WEB através do JSP(Java Server Pages).

Vale ressaltar que há muitos outros cenários em que a linguagem é aplicável e também utilizada na atualidade.

4.4 Ubiquidade

Algo que é ubíquo, representa a capacidade de estar ao mesmo tempo em toda a parte(onipresente) e difundido por todos os lados(geral, universal). A portabilidade da linguagem permitiu a ela alcançar tamanho feito, uma vez que mais de 2 bilhões de dispositivos no mundo inteiro estão rodando alguma aplicação feita em Java.

Citado este fato, é preciso então mencionar a importância da linguagem no mais novo conceito que está sendo introduzido nos tempos atuais, a "Internet das Coisas"ou IoT(Internet of Things). O IoT, segundo a própria Oracle, descreve a rede de objetos físicos—"coisas"—que são incorporados a sensores, software e outras tecnologias com o objetivo de conectar e trocar dados com outros dispositivos e sistemas pela internet.

Esses dispositivos variam de objetos domésticos comuns a ferramentas industriais sofisticadas, havendo mais de 7 bilhões de dispositivos IoT conectados atualmente. Os especialistas esperam que esse número aumente para 10 bilhões até 2020 e 22 bilhões até 2025.

5 Linguagens similares ou "confrontantes" (contraditórias)

5.1 C

C surgiu em 1972 como uma linguagem que se encontra dentro do paradigma imperativo, mais especificamente utilizando um paradigma procedural para a estruturação e execução do código fonte. Tal linguagem não permite a utilização do paradigma OO, mas tem como característica relevante um desempenho acima da média, sendo então uma das linguagens mais utilizadas após seu lançamento no desenvolvimento de diversos tipos de software em vários domínios de aplicação.

Java por sua vez, inspirando-se no fato de que C era uma linguagem que explorava fortemente os conceitos básicos do paradigma imperativo, adotou como padrão da linguagem uma estrutura caracterizada como "C-like", ou seja, as estruturas básicas de código como repetições, condicionais, tipos primitivos, etc, são extremamente parecidas com o que se observa na linguagem C.

Ao passo que Java implementa o paradigma OO, mesmo que não completamente, ele abriu as portas em 1995 para uma nova forma de desenvolver software, permitindo a utilização de conceitos de abstração mais profundos, portabilidade através da JVM e também uma curva de aprendizado tênue, comparado a que se tinha em C, tornando o seu aprendizado muito mais interessante e as aplicações muito mais simples em termos de produção.

Sendo assim, Java e C são linguagens que tem um aspecto contraditório no que se trata do paradigma que ambas focam, apesar de estarem contidas no imperativo. Por sua vez, há também similaridades, pois Java contém estruturas muito parecidas ao que se observa em C e claramente é possível observar que a linguagem desenvolvida pela Sun Microsystems implementa muitas características já vistas na linguagem criada por Dennis Ritchie no Bell Labs.

5.2 C++

Acompanhando o lançamento da primeira versão do C++ em 1985 por Bjarne Stroustrup, e posteriormente da segunda versão em 1989, o Java teve múltiplas fontes de inspiração que influenciaram o seu desenvolvimento. O fato de C++ já implementar em sua segunda versão o paradigma OO, herança, classes abstratas e métodos estáticos, mas não permitir criar um código que pudesse ser portado entre diversas plataformas sem a necessidade de uma recompilação, foi um dos gatilhos para que Java trouxesse as mesmas características do paradigma OO já vistas em C++, mas também assegurasse a possibilidade de executar um mesmo programa construído através do mesmo código fonte em plataformas distintas.

Sendo assim, é fácil dizer que C++ e Java tem similaridades, mas também são linguagens concorrentes, pois tem como principal característica tornar mais fácil o que era até então difícil, ou seja, programar em C.

Em termos de "*code debugging*", a liberdade que C++ garante para trabalhar com níveis mais baixos não é permitida facilmente em Java, uma vez que a linguagem busca reduzir a quantidade de erros gerados por manipulações incorretas/indevidas de recursos em baixo nível. A JVM tem como principal objetivo garantir que os erros possam ser facilmente identificados e corrigidos, ao passo que C++, mesmo sendo mais "liberal", sequer retorna ao usuário a razão pela qual ocorre um determinado erro, nem mesmo fornecendo um *stack trace* para auxiliar o debug, como acontece em Java.

Já acerca do desempenho de ambas as linguagens, C++, sendo um derivado de C, está mais próximo do baixo nível, permitindo acessar diretamente recursos que, em Java, são bloqueados pela JVM, a qual atua como uma interface entre o sistema operacional e o programa que está sendo executado, dificultando assim o acesso a recursos de nível mais baixo no intuito de reduzir a quantidade de erros produzidos pelos programadores. Tal característica presente em Java faz com que esta seja mais segura em relação a C++. Dentre os aspectos que influenciam o desempenho comparativo das linguagens, é possível citar o fato de C++ ser totalmente compilado e Java passar por um processo de interpretação dos bytecodes.

Vale ressaltar que em C/C++ não há um recurso automático que trabalhe em função de auxiliar o programador a gerenciar os recursos de memória alocados durante a execução dos programas, sendo

o desenvolvedor responsável por tal controle. Java por sua vez traz uma inovação que é o "*Garbage Collector*, um mecanismo automático que desaloca os recursos não mais necessários para a execução do software, dando ao programador a liberdade para se preocupar mais com a resolução do problema proposto do que com a utilização dos recursos computacionais(memória, processador, etc) disponíveis.

5.3 C#

A Microsoft, com o objetivo de criar uma linguagem que tornasse mais fácil trabalhar com estruturas complicadas presentes em C, C++ e Java, propôs uma linguagem que fosse simples, moderna e orientada a objetos. Surgiu então em 2000 o C#, cujas razões para ser utilizado eram ser uma linguagem poderosa e flexível, utilizar poucas palavras reservadas, ser modular e garantidamente popular. Sendo assim, a versão 1.0 de C# foi basicamente similar ao que Java já apresentava no momento em questão, e era claramente inferior a sua concorrente.

Através do lançamento de novas versões, o C# foi se tornando bastante familiar com o Java em diversos aspectos. Desta forma, é impossível dizer que não há similaridade entre as linguagens, pois C# carrega em si o objetivo de melhorar coisas que Java não havia conseguido realizar de forma eficiente o bastante na visão da Microsoft.

Buscando comparar as duas linguagens, há diversos domínios de aplicação em que as duas podem ser concorrentes. Vale citar que tanto Java quanto C# estão intensamente presentes no mundo dos jogos eletrônicos. Uma curiosidade interessante é que o Minecraft, feito em Java, foi adquirido enquanto propriedade da Microsoft, ao passo que o Xbox é construído em C e roda o Minecraft utilizando a plataforma XNA, por sua vez construída em sua maior parte usando C#. Há também diversos sistemas WEB que foram construídos utilizando-se JSP, mas que poderiam ser migrados para .NET utilizando C# enquanto linguagem principal, dada a capacidade de converter um software em Java para C# sem muita complexidade adicional.

5.4 Python

No início da década de 90, surgia uma linguagem que futuramente viria a disputar com o Java no quesito popularidade: Python.

Python é uma linguagem de alto nível, open source, orientada a objetos e, assim como Java, surgiu com o propósito de ser uma linguagem simples e versátil, também tendo sua sintaxe derivada, em partes, do C e pertencendo ao paradigma imperativo. Apesar de se assemelharem nesses pontos, Python possui características derivadas de Haskell, Modula-3 e Perl e é utilizado, em sua maioria, em nichos distintos dos quais Java se destaca. Java é comumente utilizado em aplicações *desktop*, sistemas embarcados, mobile, softwares empresariais, WEB, entre outros, enquanto Python ganhou fama em áreas da tecnologia que são tendências atuais, como inteligência artificial e análise de dados.

Dada a simplicidade da linguagem do Python, esta surge também como uma forte opção ao ensino e aprendizagem de programação em níveis inicias, pois a curva de aprendizado é ainda mais suave do que a presente no aprendizado de Java. Vale citar que Python está presente em um mundo onde a tipagem é fraca e a sua execução pode ser feita de forma interativa, uma vez que a linguagem é totalmente interpretada, caracterizando uma grande diferença em relação ao Java, que possui apenas parte do processo de execução ocorrendo através de interpretação.

Muito se discute sobre o fato de que Python, através de frameworks como Django, poderia ocupar o lugar do Java para WEB com o JSP. A verdade é que ambas linguagens se destacam em alguns cenários comuns, mas em sua grande maioria possuem distinções que permitem realizar uma opção por uma ou por outra de acordo com a necessidade.

5.5 Kotlin

Em 2016, os olhos dos desenvolvedores Android se abriram para observar a ascensão do Kotlin. Essa que é uma linguagem de tipagem estática, concisa e segura, segundo os próprios desenvolvedores da linguagem. Em 2017, o Google anunciou o suporte de Kotlin para Android e então deu-se início a "briga"entre as linguagens.

Desenvolvido pela JetBrains, Kotlin é uma linguagem que se encontra no domínio do paradigma imperativo. Existe a possibilidade de programar utilizando o paradigma OO, havendo também a possibilidade de utilizar o paradigma funcional.

Há algum tempo já se discute sobre Kotlin vir a ser o substituto de Java no desenvolvimento de aplicações Android nativas, que é o cenário onde ambas linguagens concorrem diretamente. Existem muitos argumentos a favor de Kotlin, principalmente utilizando-se de explicações que tentam comprovar que Java já está ultrapassado. Todavia, o cenário atual mostra que Java ainda segue forte no segmento mobile e também fora dele.

Neste sentido há um claro conflito entre as linguagens, mas que não é necessariamente acerca da sua estruturação, mas sim sobre seus melhores casos de aplicação.

5.6 PHP

Inicialmente denominada Personal Home Page Tools, tendo depois seu nome alterado para PHP: Hypertext Preprocessor ou simplesmente PHP, como é popularmente conhecida. É uma linguagem interpretada de software livre que atua principalmente no ambiente de desenvolvimento WEB, sendo responsável por comunicar a aplicação ao seu respectivo backend.

PHP é uma linguagem com fortes inspirações em C/C++, mas que abandonou a tipagem forte e adotou uma tipagem fraca (dinâmica), tornando a escrita do código mais versátil, prática e simples do ponto de vista do programador. Tal decisão garante ao programador uma possibilidade enorme de abstrair tipos, podendo então se preocupar cada vez menos com essa questão, e por consequência passa a poder ter um foco maior na resolução do problema proposto. PHP é uma linguagem que se encontra no domínio do paradigma imperativo, existindo a possibilidade de programar utilizando o paradigma procedural, e também o paradigma OO.

Em um cenário em que se discute aplicações WEB, Java e PHP entram como fortes candidatas e concorrentes no sentido de que ambas desempenham uma mesma função dentro da aplicação. Há muito tempo se discute se PHP é superior ao Java no quesito relativo ao desenvolvimento WEB, mas a medida que *frameworks* foram sendo desenvolvidos, a escolha entre uma ou outra tornou-se mais uma questão de saber qual linguagem melhor atende os requisitos que necessitam ser implementados.

6 Exemplo(s) de programa(s)

6.1 Exemplo de polimorfismo de inclusão usando herança e sobrecarga em Java

No anexo 1 há um exemplo demonstrando a utilização de herança e sobrecarga em Java. A classe Animal serve como um template para as classes Cachorro e Cobra, que herdam métodos e atributos da classe Animal. A sobrecarga acontece no método emitirSom(), permitindo que as classes filhas possam sobrepor esse método em suas respectivas implementações.

6.2 Exemplo de CRUD Genêrico - Uso de Generics

No anexo 2, existe um trecho de código que demonstra o princípio da concepção da estrutura de uma classe em Java, que tem por função ser um CRUD de tipos genéricos para trabalhar com a escrita/manipulação de arquivos de dados em memória secundária.

Logo na definição da assinatura da classe já é possível observar um tipo de abstração relacionado a herança, que é utilizada para permitir que a classe seja capaz de manipular quaisquer tipos de

variáveis que possa receber e aplicar os métodos, que também atuam sobre os tipos genéricos (sendo esse, outro tipo de abstração de tipos), sem que haja distinção dos tipos recebidos, ou seja, tendo um comportamento padrão em todas as situações.

Outra coisa interessante, é reparar na definição do método construtor `"Constructor<T> construtor;"` que foi definido genericamente sobre o tipo T, e que garante que não é preciso estar preocupado com o tipo de variável a ser manipulado pelo objeto do tipo CRUD que for utilizado, apenas sendo importante saber que ao final da execução dos métodos a ação desejada terá sido executada, independentemente de estar manipulando Registros de Usuarios, Grupos, Mensagens, Convites, etc.

6.3 Exemplo de procedimentos com sobrescrita de método

No anexo 3, existem dois métodos exemplificando o que seria uma função de retorno void (procedimento) em Java. A ideia é demonstrar que as estruturas sintáticas em Java se assemelham fortemente a C/C++. Além disso é possível ver nos trechos de implementação dos procedimentos, estruturas sintáticas de comandos comuns em decisões e loops, que transmitem bastante o fato de que Java é uma linguagem "C-like".

6.4 Exemplo de uso de HashMap

No anexo 4 há um exemplo demonstrando a utilização de HashMap na linguagem Java. O programa tem determinados alunos e cada um possui um identificador único, que funciona como chave do HashMap. É possível inserir, buscar e deletar um aluno sabendo o seu respectivo identificador.

6.5 Exemplo de recuperação de chave pelo valor utilizando Generics

No anexo 5, existe um método cujo objetivo é exemplificar uma função de recuperação de chaves de um mapa pelo seus valores utilizando Generics, sendo assim há também uma ideia de polimorfismo paramétrico sendo utilizada.

6.6 Exemplo de classe que representa um procedimento em uma LP genérica

No anexo 6, há uma classe que representa através de uma abstração o que seria um procedimento em uma LP genérica. É possível visualizar que esta classe segue o modelo padrão de uma classe em Java, e sendo assim serve como exemplo de uma classe simples na linguagem.

6.7 Exemplo de método principal onCreate na execução de um software escrito para Android

No anexo 7, há um exemplo do que seria um método de execução principal em um software nativo programado utilizando Java para Android. É possível visualizar no código em questão aspectos específicos do desenvolvimento para Android Mobile, como o acesso direto a atributos e entidades presentes em objetos da interface, além da utilização de Listeners.

7 Considerações finais

A execução deste trabalho possibilitou um entendimento maior sobre a linguagem de programação Java, tendo foco na compreensão de seus aspectos gerais, avaliando seu histórico, paradigmas pertencentes, características mais marcantes e estabelecendo um paralelo com algumas linguagens similares/-confrontantes.

Durante a escrita do relatório, pode-se compreender que Java é uma linguagem importantíssima no universo das linguagens de programação utilizadas atualmente. Sua criação foi e é fundamental em diversas áreas de aplicação, pois trouxe consigo uma ideia que ainda não havia sido explorada por outra linguagem, a portabilidade independente do hardware, uma vez que o mesmo fosse capaz de executar a JVM. Deste ponto em diante, a linguagem tomou para si a responsabilidade de ditar o que seria uma linguagem orientada a objetos, portável e que permitisse a obtenção de resultados satisfatórios, seja em desenvolvimento *desktop*, mobile ou WEB.

A Google por sua vez acabou adotando o Java enquanto linguagem principal para o desenvolvimento de seu sistema operacional voltado para dispositivos móveis em geral, o Android, e mesmo antes deste fato, o Java já era amplamente utilizado por milhares de escolas, empresas, programadores e estudantes por todo o mundo. Vale ressaltar que atualmente Java está presente em mais de 2 bilhões de dispositivos em todo o mundo, um feito nunca antes visto antes do lançamento da linguagem.

Durante o paralelo realizado com as linguagens similares/confrontantes, foi possível observar que Java está numa posição de relativa igualdade ou mesmo de igual importância em relação às linguagens que disputam diretamente o mesmo mercado ou cenário de aplicação. Sendo assim, compreende-se que Java ainda perdurará por mais alguns anos e permitirá o desenvolvimento de muitos outros sistemas e aplicações.

Há de se compreender que novas linguagens surgem com bastante frequência, mas todas elas têm como inspiração as linguagens que fizeram sucesso anteriormente e, certamente, quando mencionado o paradigma OO, Java é uma referência em termos de facilidade, praticidade, aplicação e segurança.

A Oracle vem fazendo um excelente trabalho na manutenção e atualização da linguagem, e não há como saber quais são os seus limites, pois a criatividade advém dos programadores, mas as possibilidades são garantidas pela linguagem, sendo estas "infinitas". A conclusão final então resume-se a uma única frase: "Aprenda e use Java, pois esta ainda perdurará por muitos anos".

Referências

[1] DEVELOPER. What is C#?—Developer.com. Acesso em 24 abr. 2021. Disponível em: https://www.developer.com/guides/what-is-c/.

[2] FLAPPY BIRD. Play Flappy Bird. Acesso em 24 abr. 2021. Disponível em: https://flappybird.io/.

[3] GABBRIELLI, M. , MARTINE, S. Programming Languages: Principles and Paradigms. Springer-Verlag , Londres. 2010. Acesso em 23 abr. 2021. Disponível em: http://websrv.dthu.edu.vn/attachments/newsevents/content2415/Programming_Languages_Principles_and_Paradigms_thereds1106.pdf.

[4] GOOGLE. Android – Serviços do Google Mobile. Acesso em 24 abr. 2021. Disponível em: https://www.android.com/intl/pt-BR_br/gms/.

[5] GRAND THEFT AUTO: San Andreas. Grand Theft Auto: San Andreas - Apps no Google Play. Acesso em 24 abr. 2021. Disponível em: https://play.google.com/store/apps/details?id=com.rockstargames.gtasa&hl=pt_BR&gl=US.

[6] INFO ESCOLA. C++ - Linguagem de Programação - InfoEscola. Acesso em 24 abr. 2021. Disponível em: https://www.infoescola.com/informatica/cpp/.

[7] JAVA. Java — Oracle. Acesso em 24 abr. 2021. Disponível em: https://www.java.com/pt-BR/.

[8] KOTLIN PROGRAMMING LANGUAGE. A modern programming language that makes developers happier. Acesso em 24 abr. 2021. Disponível em: https://kotlinlang.org/.

[9] MICROSOFT DOCS. O histórico da linguagem C# - Guia do C# — Microsoft Docs. Acesso em 24 abr. 2021. Disponível em: https://docs.microsoft.com/pt-br/dotnet/csharp/whats-new/csharp-version-history.

[10] MICROSOFT. Microsoft - Official Home Page. Acesso em 24 abr. 2021. Disponível em: https://www.microsoft.com/pt-br.

[11] MICROSOFT. Download Microsoft XNA Game Studio 4.0 from Official Microsoft Download Center. Acesso em 24 abr. 2021. Disponível em: https://www.microsoft.com/en-us/download/details.aspx?id=23714.

[12] MICROSOFT. Documentação do C++ – introdução, tutoriais, referência. — Microsoft Docs. Acesso em 24 abr. 2021. Disponível em: https://docs.microsoft.com/pt-br/cpp/cpp/?view=msvc-160.

[13] MINECRAFT. Site oficial — Minecraft. Acesso em 24 abr. 2021. Disponível em: https://www.minecraft.net/pt-br/.

[14] MOVILE. Kotlin vs Java: motivos para trocar o Java pelo Kotlin ainda hoje. Acesso em 24 abr. 2021. Disponível em: https://movile.blog/motivosparatrocarojavapelokotlinaindahoje/.

[15] ORACLE. Java.util.fuction. Acesso em 23 abr. 2021. Disponível em: https://docs.oracle.com/javase/8/docs/api/java/util/function/packagesummary.html.

[16] ORACLE. JavaServer Pages Technology. Acesso em 24 abr. 2021. Disponível em: https://www.oracle.com/java/technologies/jspt.html.

[17] ORACLE. Applets. Acesso em 24 abr. 2021. Disponível em: https://www.oracle.com/java/technologies/applets.html.

[18] ORACLE. O Que é Internet of Things (IoT)? — Oracle Brasil. Acesso em 24 abr. 2021. Disponível em: https://www.oracle.com/br/internetofthings/whatisiot/.

[19] ORACLE. Our world. Moved by Java. Acesso em 24 abr. 2021. Disponível em: https://www.oracle.com/java/movedby-java/timeline/?source=:ow:o:p:po::RC_WWMK200728P00026:DevLiveJava_PromoOcomJava&intcmp=:ow:o:p:po::RC_WWMK200728P00026:DevLiveJava_PromoOcomJava#2015.

[20] ORACLE. Oracle anuncia o Java 15. Acesso em 24 abr. 2021. Disponível em: https://blogs.oracle.com/oracle-brasil/oracle-anuncia-java-15#:~text=o%20JEP%20371%3A%20Hidden%20Classes,as%20usam %20indiretamente%2C%20via %20reflex%C3%A3o.&text=Classes%20e%20interfaces%20seladas%20restringem,%2Dlas%20ou%20implement%C3%A1%2Dlas.

[21] RUNESCAPE. Comunidade On-line do RuneScape - Fóruns, Notícias, Eventos e muito mais. Acesso em 24 abr. 2021. Disponível em: https://www.runescape.com/community.

[22] SILVA, F. Programação Orientada a Objetos. Universidade Federal de Uberlândia. Uberlândia, Brasil. 2004. Acesso em 23 abr. 2021. Disponível em: http://www.facom.ufu.br/~flavio/poo/files/2004-01/Pootad.pdf.

[23] SINGH, K. , IANCULESCU,A. , TORJE, L. Design Patterns and Best Practices in Java. 2018. Acesso em 23 abr. 2021. Disponível em: https://subscription.packtpub.com/book/application_development/9781786463593/1/ch01lvl1sec13/objectorientedparadigm.

[24] STARTCOUNTER GLOBAL STATS. Mobile Operating System Market Share Worldwide — StatCounter Global Stats. Acesso em 24 abr. 2021. Disponível em: https://gs.statcounter.com/os-market-share/mobile/worldwide.

[25] TED. TED: Ideas worth spreading. Acesso em 28 abr. 2021. Disponível em: https://www.ted.com/.

[26] TIBIA. Tibia - Free Massively Multiplayer Online Role-Playing Game - MMORPG. Acesso em 24 abr. 2021. Disponível em: https://www.tibia.com/mmorpg/free-multiplayer-online-role-playing-game.php.

[27] THE SIMS. Jogos The Sims - Site Oficial da EA. Acesso em 24 abr. 2021. Disponível em: https://www.ea.com/pt-br/games/the-sims.

[28] WEISFELD, M. The Object-Oriented Thought Process. 2009. Acesso em 23 abr. 2021. Disponível em: https://www.researchgate.net/publication/262350549_The_object-oriented_thought_process_fourth_edition_by_Matt_Weisfeld.

8 Anexos

8.1 Anexo 1

```
1  import java.io.*;
2
3  // classe generica para definir um animal
4  class Animal{
5      // atributos protegidos para que apenas os filhos,
           alem da propria classe pai possam acessar
6      protected String nome;
7      protected String classe;
8
9      // contrutor da classe Animal
10     Animal(String nome, String classe){
11         this.nome = nome;
12         this.classe = classe;
13     }
14
15     // getters e setters para recuperar ou definir
           valores de atributos de objetos do tipo Animal
16     String getNome(){
17         return nome;
18     }
19
20     void setNome(String nome){
21         this.nome = nome;
22     }
23
24     String getClasse(){
25         return classe;
26     }
27
28     void setClasse(String classe){
29         this.classe = classe;
30     }
31
32     void emitirSom(){
33
34     }
35
36 }
37
38 //Cachorro herda os atributos e metodos publicos/
       protegidos de Animal
39 class Cachorro extends Animal{
40     // novos atributos sao declarados para especificar um
           cachorro
41     private String corPelo;
42     private String raca;
43
44     Cachorro(String nome, String classe, String corPelo,
           String raca){
45         super(nome, classe);
46         this.corPelo = corPelo;
47         this.raca = raca;
```

```
48      }
49
50      String getCorPelo(){
51          return corPelo;
52      }
53
54      void setCorPelo(String corPelo){
55          this.corPelo = corPelo;
56      }
57
58      String getRaca(){
59          return raca;
60      }
61
62      void setRaca(String raca){
63          this.raca = raca;
64      }
65
66      void mostrar(){
67          System.out.println("Nome: " + nome + "\nClasse: "
                + classe + "\nCor do Pelo" + corPelo + "\
               nRaca: " + raca);
68      }
69
70      // metodo utilizado apenas por objetos do tipo
           Cachorro
71      @Override
72      void emitirSom() {
73          // TODO Auto-generated method stub
74          System.out.println("Au au...");
75      }
76  }
77
78  //Cobra herda os atributos e metodos publicos/protegidos
       de Animal
79  class Cobra extends Animal{
80      // novos atributos sao declarados para especificar
           uma cobra
81      private boolean veneno;
82      private String especie;
83
84      Cobra(String nome, String classe, boolean veneno,
           String especie){
85          super(nome, classe);
86          this.veneno = veneno;
87          this.especie = especie;
88      }
89
90      boolean getVeneno(){
91          return veneno;
92      }
93
94      void setVeneno(boolean veneno){
95          this.veneno = veneno;
96      }
97
```

```
98      String getEspecie(){
99          return especie;
100     }
101
102     void setEspecie(String especie){
103         this.especie = especie;
104     }
105
106     @Override
107     void emitirSom() {
108         // TODO Auto-generated method stub
109         System.out.println("Sssssss...");
110     }
111
112     void mostrar(){
113         if(veneno){
114             System.out.println("Nome: " + nome + "\
                    nClasse: " + classe + "\nVenenosa" + "\
                    nEspecie: " + especie);
115         } else {
116             System.out.println("Nome: " + nome + "\
                    nClasse: " + classe + "\nNao venenosa" + "
                    \nEspecie: " + especie);
117         }
118     }
119 }
120
121 public class exemploClasseHeranca{
122     public static void main(String[] args){
123         Cachorro cachorro = new Cachorro("Bob", "Mamifero
                ", "preto", "Shih-tzu");
124         cachorro.mostrar();
125         cachorro.emitirSom();
126         Cobra cobra = new Cobra("Pablo", "Reptil", false
                ,"Python");
127         cobra.mostrar();
128         cobra.emitirSom();
129     }
130 }
```

Listing 4: Exemplo de polimorfismo de inclusão usando herança e sobrecarga em Java

8.2 Anexo 2

```
1  package amigooculto.bancoDeDados;
2
3  //Importacoes
4  import amigooculto.indices.ArvoreBMais_String_Int;
5  import amigooculto.indices.HashExtensivel;
6  import amigooculto.interfaces.Registro;
7  import java.io.File;
8  import java.io.IOException;
9  import java.io.RandomAccessFile;
10 import java.lang.reflect.Constructor;
11
12 /**
```

```
13  * Classe CRUD
14  * Feita de forma gen rica
15  * @author Jonathan
16  * @param <T>
17  */
18 public class CRUD<T extends Registro> {
19
20     //Atributos da classe
21     Constructor<T> construtor;
22     //nome do diretorio onde ser o armazenados os dados
23     public final String diretorio = "dados";
24
25     //vari vel de acesso aos arquivos permitindo escrita
           e leitura
26     private final RandomAccessFile arquivo;
27     //  ndices
28     private final HashExtensivel indiceDireto;
29     private final ArvoreBMais_String_Int indiceIndireto;
30
31
32     /**
33      * Construtor da Classe CRUD
34      * @param nomeArquivo
35      * @param construtor
36      * @throws Exception
37      */
38     public CRUD(String nomeArquivo, Constructor<T>
          construtor) throws Exception {
39         this.construtor = construtor; //inicia o
              construtor gen rico
40
41         //abre o arquivo no diretorio especificado
42         File d = new File(this.diretorio);
43         //verifica se j  existe o diret rio, sen o
              cria
44         if (!d.exists()) {
45             d.mkdir();
46         }
47
48         //abre o arquivo em modo leitura e escrita
49         arquivo = new RandomAccessFile(this.diretorio + "
              /" + nomeArquivo + ".db", "rw");
50
51         //caso o tamanho do arquivo seja inferior a 4
              bytes escreve o cabe alho do arquivo com
52         //um inteiro de valor 0
53         if (arquivo.length() < 4) {
54             arquivo.writeInt(0);  // cabe alho do
                  arquivo
55         }
56
57         //inicializa os  ndices
58         indiceDireto = new HashExtensivel(10,
59                 this.diretorio + "/diretorio." +
                      nomeArquivo + ".idx",
60                 this.diretorio + "/cestos." + nomeArquivo
```

```
                    + ".idx");

        indiceIndireto = new ArvoreBMais_String_Int(10,
                this.diretorio + "/indiceIndireto." +
                    nomeArquivo + ".idx");
    }

    //M todos de utiliza  o do CRUD
    public int create(T novaInstancia) throws IOException
        , Exception {
        arquivo.seek(0); //posiciona o ponteiro no inicio
            do arquivo
        int ultimoId = arquivo.readInt(); //obtem o
           ultimo id gravado
        ultimoId++; //incrementa o id
        arquivo.seek(0); //reposiciona o ponteiro no
           inicio
        arquivo.writeInt(ultimoId); //grava o ultimo id
           registrado

        novaInstancia.setId(ultimoId); //cria nova
           inst ncia com o valor do  ltimo  id
           registrado
        long enderecoInsercao = arquivo.length(); //marca
            o endere o de inser  o
        arquivo.seek(enderecoInsercao); //posiciona o
           ponteiro no endere o de inser  o
        arquivo.writeBoolean(true); //escreve a l pide
        arquivo.writeInt(novaInstancia.toByteArray().
           length); //escreve o indicador de tamanho do
           registro
        arquivo.write(novaInstancia.toByteArray()); //
           escreve os dados do registro

        indiceDireto.create(ultimoId, enderecoInsercao);
           //registra no  ndice  direto a posi  o em
           que se encontra o registro adicioando
        indiceIndireto.create(novaInstancia.
           chaveSecundaria(), ultimoId); //registra no
            ndice  indireto

        return novaInstancia.getId(); //retorna o id do
           registro no arquivo
    }

    public T read(int id) throws Exception {
        long endereco = indiceDireto.read(id); //obt m o
            endere o do registro no arquivo
        //System.out.println("EnderecoREADID = " +
           endereco);

        //verifica a n o exist ncia desse endere o
        if (endereco == -1) {
            return null;
        }
```

```
96          arquivo.seek(endereco); //posiciona o ponteiro no
                endere o para leitura
97          boolean lapide = arquivo.readBoolean(); //l   a
               l pide
98
99          //verifica se a l pide    v lida
100         if (!lapide) {
101             return null;
102         }
103
104         //obtem o indicador de tamanho do registro
105         int tamanho = arquivo.readInt();
106         byte[] array = new byte[tamanho];
107
108         arquivo.read(array); //l   o array bytes do
               arquivo (dados do registro)
109         T instancia = this.construtor.newInstance(); //
               cria de forma gen rica o objeto
110         instancia.fromByteArray(array); //introduz os
               dados no objeto
111         instancia.setId(id); //seta o id
112
113         //retorna a inst ncia recuperada
114         return instancia;
115     }
116
117     /**
118      * Faz a mesma coisa do read acima s   que utiliza a
            chave secundaria para obter
119      * o id que ser  utilizado para realizar a leitura
120      * @param email
121      * @return
122      * @throws IOException
123      * @throws Exception
124      */
125     public T read(String email) throws IOException,
           Exception {
126         int id = indiceIndireto.read(email);
127         return read(id);
128     }
129
130     public boolean update(T novaInstancia) throws
           Exception {
131         T instanciaAntiga = read(novaInstancia.getId());
               //recupera a inst ncia antiga a partir de seu
                id
132         long endereco = indiceDireto.read(instanciaAntiga
               .getId()); //obt m o endere o do registro no
                arquivo
133         arquivo.seek(endereco + 5); //pula o cabe alho
134         int tamanhoAtual = novaInstancia.toByteArray().
               length; //tamanho do novo registro
135         int tamanhoAnterior = instanciaAntiga.toByteArray
               ().length; //tamanho anterior do registro
136
137         //trecho de verifica  o da necessidade de
```

```
            reescrever o registro numa nova posi  o
138         //caso tenha aumentado de tamanho e o escreve na
               mesma posi  o caso tamanho seja
139         //menor ou igual
140         if (tamanhoAtual <= tamanhoAnterior) {
141             arquivo.write(novaInstancia.toByteArray());
142         } else {
143             arquivo.seek(endereco);
144             arquivo.writeBoolean(false);
145             long enderecoInsercao = arquivo.length();
146             arquivo.seek(enderecoInsercao);
147             byte[] array = novaInstancia.toByteArray();
148             arquivo.writeBoolean(true);
149             arquivo.writeInt(array.length);
150             arquivo.write(array);
151             indiceDireto.update(novaInstancia.getId(),
                   enderecoInsercao);
152         }
153
154         //atualiza o  ndice  indireto em caso de mudan a
155         if (!instanciaAntiga.chaveSecundaria().equals(
               novaInstancia.chaveSecundaria())) {
156             indiceIndireto.delete(instanciaAntiga.
                   chaveSecundaria());
157             indiceIndireto.create(novaInstancia.
                   chaveSecundaria(), novaInstancia.getId());
158         }
159
160         return true;
161     }
162
163     public boolean delete(int id) throws Exception {
164         T instancia = read(id);
165         long enderecoDelecao = indiceDireto.read(id);
166         arquivo.seek(enderecoDelecao);
167         arquivo.writeBoolean(false); //seta a l pide
               como falso invalidando o registro
168         //faz as dele  es necess rias nos  ndices
169         indiceIndireto.delete(instancia.chaveSecundaria()
               );
170         indiceDireto.delete(id);
171
172         return true;
173     }
174
175     //getter e setter
176     public String getDiretorio() {
177         return diretorio;
178     }
179 }
```

Listing 5: CRUD Genêrico - Uso de Generics

8.3 Anexo 3

```
@Override
    public void onAction(String binding, boolean value,
       float tpf) {
        if (binding.equals("Destroy")) {
            floor_geo.setMaterial(floor_mat);
            floor_geo.setLocalTranslation(0, -0.4f, 0);
            this.rootNode.attachChild(floor_geo);
            /* Make the floor physical with mass 0.0f! */
            floor_phy = new RigidBodyControl(1.0f);
            floor_geo.addControl(floor_phy);
            bulletAppState.getPhysicsSpace().add(
               floor_phy);
            floor_phy.setRestitution(1f);

            /**
             * Position the brick geometry
             */
            brick_geo.setLocalTranslation(location_1);
            brick_geo2.setLocalTranslation(location_2);
            brick_geo3.setLocalTranslation(location_3);
            brick_geo4.setLocalTranslation(location_4);
            brick_geo5.setLocalTranslation(location_5);
            brick_phy = new RigidBodyControl(1.0f);
            brick_phy2 = new RigidBodyControl(1.0f);
            brick_phy3 = new RigidBodyControl(1.0f);
            brick_phy4 = new RigidBodyControl(1.0f);
            brick_phy5 = new RigidBodyControl(1.0f);

            brick_geo.addControl(brick_phy);
            brick_geo2.addControl(brick_phy2);
            brick_geo3.addControl(brick_phy3);
            brick_geo4.addControl(brick_phy4);
            brick_geo5.addControl(brick_phy5);
            bulletAppState.getPhysicsSpace().add(
               brick_phy);
            bulletAppState.getPhysicsSpace().add(
               brick_phy2);
            bulletAppState.getPhysicsSpace().add(
               brick_phy3);
            bulletAppState.getPhysicsSpace().add(
               brick_phy4);
            bulletAppState.getPhysicsSpace().add(
               brick_phy5);
            brick_phy.setRestitution(0.5f);
            brick_phy2.setRestitution(0.5f);
            brick_phy3.setRestitution(0.5f);
            brick_phy4.setRestitution(0.5f);
            brick_phy5.setRestitution(0.5f);
            points += points;
            setUpHUDText();
        }
        if (binding.equals("Shoot")) {

            Vector3f vec = new Vector3f(0f, 5f, -5f);
//
```

```
49 //            Vector3f vec1 = new Vector3f(0f, 0f, 0f);
50 //            if (ball_phy.getGravity().z == 0){
51 //                ball_phy.applyImpulse(vec,
       viewDirection);
52 //            }else{
53 //                ball_phy.setLinearVelocity(vec1);
54 //                ball_phy.setGravity(vec1);
55 //            }
56             ball_phy.applyImpulse(vec, viewDirection);
57 
58             points += 50;
59             setUpHUDText();
60             //makeCannonBall(new Vector3f(-4, 1, 0), new
                  Vector3f(6, 4, 0), fast_mat,
                  continuouslySweeping); //slow arcing ball
61         }
62         if (binding.equals("Strafe Left")) {
63             leftStrafe = value;
64         } else if (binding.equals("Strafe Right")) {
65             rightStrafe = value;
66         } else if (binding.equals("Rotate Left")) {
67             leftRotate = value;
68         } else if (binding.equals("Rotate Right")) {
69             rightRotate = value;
70         } else if (binding.equals("Walk Forward")) {
71             forward = value;
72         } else if (binding.equals("Walk Backward")) {
73             backward = value;
74         } else if (binding.equals("Jump")) {
75             physicsCharacter.jump();
76             Vector3f force = new Vector3f(0, 3f, 0);
77             ball_phy.applyImpulse(force, walkDirection);
78         }
79     }
80 
81     private PhysicsSpace getPhysicsSpace() {
82         return bulletAppState.getPhysicsSpace();
83     }
84 
85     /**
86      * COLLISION LOGIC
87      */
88 
89     public void collision() {
90         if (ghost.getOverlappingObjects().indexOf(
              ball_phy) != (-1)){
91             Vector3f vec = new Vector3f(0f, 5f, -5f);
92 //
93 //            Vector3f vec1 = new Vector3f(0f, 0f, 0f);
94 //            if (ball_phy.getGravity().z == 0){
95 //                ball_phy.applyImpulse(vec,
       viewDirection);
96 //            }else{
97 //                ball_phy.setLinearVelocity(vec1);
98 //                ball_phy.setGravity(vec1);
99 //            }
```

```
100             ball_phy.applyImpulse(vec, viewDirection);
101
102             points += 50;
103             setUpHUDText();
104         }
105     }
```

Listing 6: Exemplo de procedimentos com sobrescrita de método utilizando paradigma orientado a eventos

8.4 Anexo 4

```
1  import java.util.HashMap;
2  import java.util.Map;
3  import java.util.Iterator;
4  import java.util.Set;
5  public class HashMapExample{
6
7     public static void HashMapExample(String args[]) {
8
9        /*Declara  o de um HashMap*/
10       HashMap<Integer, String> aluno = new HashMap<
            Integer, String>();
11
12       /*Inser  o de um elemento no HashMap*/
13       aluno.put(1, "Stephanie Silva");
14       aluno.put(2, "Gustavo Gomes");
15       aluno.put(3, "Giulia Chiucchi");
16       aluno.put(4, "Eduardo Pereira");
17       aluno.put(5, "Jonathan Douglas");
18
19       /*Exibe o conteudo ho HashMap usando um iterator*/
20       Set set = aluno.entrySet();
21       Iterator iterator = set.iterator();
22       while(iterator.hasNext()) {
23          Map.Entry mentry = (Map.Entry)iterator.next();
24          System.out.print("ID: "+ mentry.getKey() + " --
               Nome: ");
25          System.out.println(mentry.getValue());
26       }
27
28       /*Resgate de valores pela chave*/
29       String aluno2 = aluno.get(2);
30       System.out.println("Aluno com o ID 2   : "+ aluno2)
            ;
31
32       /*Remove valores baseado na chave*/
33       aluno.remove(3);
34       System.out.println("Alunos depois da remo  o do
            aluno com ID 3:");
35       Set set2 = aluno.entrySet();
36       Iterator iterator2 = set2.iterator();
37       while(iterator2.hasNext()) {
38           Map.Entry mentry2 = (Map.Entry)iterator2.next()
               ;
```

```
39            System.out.print("ID: "+ mentry2.getKey() + "
                 -- Nome: ");
40            System.out.println(mentry2.getValue());
41        }
42
43    }
44 }
```

Listing 7: Exemplo de uso de HashMap

8.5 Anexo 5

```
1  public static <T, E> T getKeyByValue(Map<T, E> map, E
       value) {
2          for (Map.Entry<T, E> entry : map.entrySet()) {
3              if (Objects.equals(value, entry.getValue()))
                   {
4                  return entry.getKey();
5              }
6          }
7          return null;
8  }
```

Listing 8: Exemplo de recuperação de chave pelo valor utilizando Generics

8.6 Anexo 6

```
1  package Proc;
2
3  import Variavel.*;
4  import Comando.*;
5  import java.util.*;
6
7
8  public class Procedimento {
9    String nome;                          // nome do
       procedimento
10   Vector comandos;                        // vetor de
       comandos
11   String [] parametros;                   // parametros
       formais
12   String [] variaveisLocal;                 // variveis
       local
13   static Memoria global = new Memoria();      //
       variaveis globais
14
15   public Procedimento( String n, String [] p ){
16     nome= n;
17     parametros= p;
18   }
19
20   public String getNome(){
21     return nome;
22   }
23
24   public static void setVariaveisGlobal( String [] vG ){
```

```
25      int i = 0;
26      for (String var : vG){
27        if (var == null){break;}
28        int pos = (int)(var.charAt(0)) - 97;
29        global.var[pos] = 0.0;
30        //System.out.println("oi");
31      }
32    }
33
34    public void setVariaveisLocal( String [] vL ){
35      variaveisLocal = vL;
36    }
37
38    public void setListaComandos( Vector c ){
39      comandos = c;
40    }
41
42    public void executa(double [] argumentos) {
43      Memoria local = new Memoria();         // variaveis
           local
44      //char var;
45      int i = 0;
46      // adiciona variveis locais na memria local
47      if( variaveisLocal != null ){
48        for (String var : variaveisLocal){
49          if (var == null){break;}
50          int pos = (int)(var.charAt(0)) - 97;
51          local.var[pos] = 0.0;
52          //System.out.printf("variavelLocal[%d] = %d",pos,
                local.var[pos]);
53        }
54      }
55
56      // associa argumentos a seus respectivos par metros
           formais
57      for (String var : parametros){
58        if (var == null){break;}
59        int pos = (int)(var.charAt(0)) - 97;
60        local.var[pos] = argumentos[i];
61        i++;
62        //System.out.printf("argumentos[%d] = %d");
63      }
64
65      int pc= 0;
66      do {
67        //System.out.println("hello");
68        pc=  ((Comando) comandos.elementAt(pc)).executa(
             local, global);
69      } while (pc < comandos.size() );
70    }
71  }
```

Listing 9: Exemplo de classe que representa um procedimento em uma LP genérica

8.7 Anexo 7

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_jogo_adivinha);
        numeroEscolhido = findViewById(R.id.
           editText_numeroEscolhido);
        dicaReal = findViewById(R.id.textView_dicaReal);
        tentativasReal = findViewById(R.id.
           textView_tentativasReal);
        numeroGerado = findViewById(R.id.
           textView_numeroGerado);
        gerarNovo = findViewById(R.id.botao_gerarNovo);
        chutar = findViewById(R.id.botao_chutar);

        gerarNovo.setOnClickListener(new View.
           OnClickListener() {
            @Override
            public void onClick(View v) {
                if (!comecouAgora){
                    numeroGerado.setText(String.valueOf(
                       numeroSorteado));
                    numeroSorteado = numeroAleatorio(1,
                       50);
                    tentativas = 0;
                    dicaReal.setText("");
                    tentativasReal.setText(String.valueOf
                       (tentativas));
                    numeroGerado.setText("");
                    Toast.makeText(getApplicationContext
                       (), "Novo n mero sorteado!",
                       Toast.LENGTH_SHORT).show();
                } else {
                    Toast.makeText(getApplicationContext
                       (), "Voc  come ou agora, tente
                       adivinhar pelo menos uma vez!",
                       Toast.LENGTH_SHORT).show();
                    comecouAgora = false;
                }
            }
        });

        chutar.setOnClickListener(new View.
           OnClickListener() {
            public void onClick(View v) {
                try{
                    numeroChutado = Integer.parseInt(
                       numeroEscolhido.getText().toString
                       ());

                    if (numeroChutado < 1 ||
                       numeroChutado > 50) {
                        throw new Exception();
                    } else {
                        if (numeroChutado ==
                           numeroSorteado){
                            dicaReal.setText("");
```

```
39                                  numeroGerado.setText(String.
                                        valueOf(numeroSorteado));
40                                  Toast.makeText(
                                        getApplicationContext(), "
                                        Parab ns , voc  acertou!"
                                        , Toast.LENGTH_SHORT).show
                                        ();
41                              } else if (numeroChutado <
                                    numeroSorteado){
42                                  dicaReal.setText("Chute mais
                                        alto!");
43                              } else if (numeroChutado >
                                    numeroSorteado){
44                                  dicaReal.setText("Chute mais
                                        baixo!");
45                              }
46
47                              tentativas++;
48                              tentativasReal.setText(String.
                                    valueOf(tentativas));
49
50                              if(comecouAgora){
51                                  comecouAgora = false;
52                              }
53                          }
54                      } catch (Exception e){
55                          Toast.makeText(getApplicationContext
                                (), "Voc  inseriu valores
                                inv lidos. Insira corretamente.",
                                 Toast.LENGTH_SHORT).show();
56                      }
57                  }
58              });
59      }
```

Listing 10: Exemplo de método principal onCreate na execução de um software escrito em Java para Android utilizando o paradigma orientado a eventos

9 Apêndice - Eduardo

A realização do trabalho me permitiu descobrir características que eu não conhecia sobre o Java, como o, ainda em implementação, suporte à programação funcional e alguns conceitos históricos sobre a linguagem.

Tendo sido o Java uma das primeiras/únicas linguagens de programação com a qual tive contato, me chamou a atenção o tópico em que foi realizado uma comparação entre a mesma e outras linguagens conhecidas do meio. Ter uma noção das vantagens e desvantagens de uma linguagem sobre a outra é algo crucial para que se possa desenvolver, de forma esperta e eficiente, qualquer software.

Outro ponto que me despertou interesse foi ter adquirido o conhecimento quanto ao uso de Java na área de IoT. Saber que a linguagem na qual possuo maior domínio (no momento) é amplamente utilizada em tal área, a qual também me atrai, me fez ter uma motivação a mais em continuar sempre aprendendo coisas novas sobre Java.

Aproveitando o gancho quanto ao uso na área de IoT, a portabilidade e a flexibilidade que Java possui é algo que me encanta, uma vez que, para quem pensa em iniciar na área de desenvolvimento, é possível aprender a linguagem antes mesmo de decidir para qual escopo gostaria de desenvolver, acelerando o processo de aprendizado e de entrada no mercado. Não apenas se restringindo à iniciantes, desenvolvedores experientes podem também mudar facilmente sua área de foco, visto que a forma de se programar em Java é quase a mesma, independente se a área para qual você desenvolve é focada em sites WEB ou na criação de jogos.

Java foi e é uma das linguagens mais utilizadas no mundo e ver que uma linguagem "antiga"continua mantendo tal feito, mesmo em meio à linguagens que vieram com a especulação de que iriam "destroná-la", é algo no mínimo curioso. O trabalho foi importante para que eu pudesse entender o porquê desse grande número de utilizadores e o motivo pelo qual Java ainda mantém seu posto frente a outras linguagens.

Concluindo o resumo sobre meus conhecimentos adquiridos, Java é uma linguagem que me encanta cada vez mais. Mesmo possuindo um problema ou outro (assim como toda e qualquer linguagem), sinto que o Java ainda não mostrou todo seu potencial e que, aliado a atualização constantes de sua mantenedora, a Oracle, Java perdurará por mais alguns anos, continuando a influenciar em novas linguagens e na forma com a qual se desenvolve software.

10 Apêndice - Gustavo

Com o desenvolver do trabalho realizado no presente documento, pude aprender muito mais do que eu já sabia da linguagem tema do nosso projeto, o Java, uma linguagem que eu considero ser a que mais domino visto que foi a que mais utilizei até hoje.

Dentre todo o conteúdo da pesquisa, não poderia escolher uma parte que mais me chamou a atenção, pois, Java é uma linguagem que eu realmente me interesso, portanto, posso começar falando sobre os paradigmas, pois, antes da pesquisa ser realizada, eu apenas sabia que a linguagem pertencia ao Paradigma Orientado a Objetos, ao menos, por nomenclatura, este era o único paradigma que eu tinha noção de Java estava inserido, porém, ao ler sobre o que é o paradigma imperativo, percebi que eu já sabia que o Java estava inserido nele, só não sabia o que era o paradigma em si.

Já o paradigma declarativo, eu não fazia ideia de que Java estava inserido nele e nem o que ele era, durante a pesquisa pude aprender isso que é um conceito mais geral na programação, então, meu conhecimento não foi expandido apenas em relação ao Java, mas à programação no geral também.

Nunca tinha visto a utilização de um Stream como no código abaixo e nem sabia que Java possuía características de paradigma funcional.

```
1 IntStream.range(0, 10).filter(i -> i % 2 == 0).
2 forEach(System.out::println);
```

Listing 11: Uso de Streams

Um pouco não ortodoxo falar da história depois dos paradigmas porém, é uma parte realmente importante também, pois é possível entender como e porquê o Java foi criado, além de também citar algumas empresas que utilizam o Java em seus produtos, e que eu utilizo muito e não tinha ideia de que possuía Java "por baixo dos panos", como por exemplo, o Spotify e a Netflix.

Além disso, também achei interessante a parte das linguagens semelhantes, pois pude perceber semelhanças e diferenças que eu não havia percebido antes em algumas dessas linguagens que eu utilizava e também em outras que eu nunca tive contato como Python.

Também é curioso ver que mesmo com tanto tempo da existência do Java, ele continua sendo uma das linguagens mais utilizadas no mundo e mesmo tendo sido lançado há 26 anos ele continua crescendo e constantemente recebendo atualizações com recursos novos e até hoje mesmo com algumas linguagens sendo consideradas aquelas que iriam "acabar" com o Java ele continua "intacto".

Ademais, é curiosa a característica da ubiquidade do Java, dando a ele a possibilidade de estar sendo utilizado em tantas áreas diferentes como, jogos, sistemas operacionais de celulares, aplicativos mobile e plataformas Web e garantindo assim sua estabilidade no mercado da tecnologia.

Outro ponto, digamos, marcante do documento são os exemplos, que me permitiram ver como o Java pode ser aplicado em diferentes resoluções de problemas, como é o caso de sua utilização em um destes para a criação de uma aplicação Android de um jogo simples de adivinhação.

De modo geral, a realização deste trabalho foi um grande adendo ao meu conhecimento, que agora, percebo que não era tão profundo na "parte teórica" da linguagem e mais focado apenas na prática que é onde eu mais me foco em qualquer linguagem ou tecnologia. O desenvolvimento deste trabalho, posso dizer que, abriu meus olhos para pesquisar mais além da parte prática de uma linguagem, pois conhecendo os conceitos e definições de paradigmas, características, entre outros, fica bem mais fácil de perceber esses "detalhes" na prática.

11 Apêndice - Jonathan

Me considero desde 2013 um desenvolvedor Java, pois desde o meu curso Técnico em Informática pelo CEFET-MG tive contato com a linguagem, sendo naquele tempo na versão do Java 7. Meu conhecimento acerca de POO era escasso e a medida que fui aprendendo mais sobre OO, pude me tornar capaz de explorar o Java de uma forma muito mais interessante e intensa.

Atualmente, através deste trabalho e de outras experiências que tive com a linguagem pude perceber que a minha paixão por Java não é atoa, ele realmente é uma linguagem sensacional. O desenvolvimento deste trabalho me permitiu conhecer ainda mais sobre a história de uma linguagem que eu aprendi a amar, que utiliza um paradigma que eu aprendi a valorizar e possui um pé em outros paradigmas que eu só fui descobrir que existiam anos depois do meu primeiro contato com a linguagem.

Tendo aprendido mais sobre a história, consegui também saber mais sobre suas características técnicas, *features*, propósitos, áreas de aplicação, cenários em que se destaca e com isso aprender mais sobre suas influências e concorrentes diretas e indiretas.

Se manter no mercado por tanto tempo, tornando-se a escolha da Google para o sistema operacional Android, a principal linguagem utilizada por muitos dos serviços mais conhecidos no mundo em diversas áreas e por ter sua característica ubíqua, além de ser uma linguagem que dita os novos paradigmas da IoT(Internet of Things) não é para muitas linguagem, na verdade diria que esse título só pertence ao Java e consequentemente a Oracle. Não é mera coincidência o fato de que Java está em constante evolução, pois a Oracle entende bem o seu poder e o quão longe essa linguagem ainda pode ir.

Uma das razões pela qual sou tão apaixonado pela linguagem e porque esse trabalho foi tão interessante para mim deve-se ao fato de que Java é uma linguagem que me permite explorar de forma ampla toda a minha capacidade de pensamento através da abstração. A forma com que Java implementa o POO me permite ir além do que eu já fazia em linguagens como C, C++, PHP e outras de suas influências e concorrentes com as quais já tive contato.

Pode parecer que estou puxando a sardinha para o Java pelo simples fato de eu gostar muito da linguagem, mas é preciso reconhecer sua importância e seu impacto na idealização das linguagens posteriores a ela, pois ela alcançou o poder de ditar padrões acerca de como as linguagens contemporâneas devem lidar com temas como abstração, curva de aprendizado tênue, documentação, portabilidade e ubiquidade, que é uma consequência direta da proposta introduzida pela Sun, "Write Once Run Anywhere".

Não posso afirmar com certeza que sei até onde essa linguagem irá, mas ela com certeza não será substituída e não morrerá como já fora cravado diversas vezes por aí. Java é uma linguagem que atualmente, em minha singela opinião, deve ser aprendida e utilizada quando possível por todos os desenvolvedores, pois ela é referência em diversas áreas como portabilidade, segurança, flexibilidade e facilidade de uso e aprendizado. E finalizando minha contribuição, vale a pena citar o fato de que Java foi a única e principal linguagem que utilizei assim que ingressei na Ciência da Computação na PUC-MG em 2015, em que por dois períodos utilizei apenas Java nas disciplinas de AEDS1, AEDS2 e Matemática Discreta.

12 Apêndice - Stephanie

Durante a realização do trabalho consegui conhecer mais a fundo uma das linguagens na qual tenho mais prática, o Java. Um dos pontos que mais me chamou a atenção foi os paradigmas no qual a linguagem está presente, uma vez que não sabia da possibilidade de Java também possuir características do paradigma funcional.

Desse modo, é possível trabalhar os melhores aspectos que cada paradigma oferece dentro da linguagem e até mesmo combiná-los.Além disso, ficou evidenciado que Java é uma referência no paradigma orientado a objetos e outras linguagens que surgiram após a mesma, como C, se espelham bastante em Java para definir características do POO.

Outro ponto interessante abordado no documento é a diversidade de segmentos no qual Java está presente. Variando do setor bancário, educacional, tecnológico, e até o de games. A linguagem tem o poder de ser adaptável e portátil para diversos cenários e situações. Está ligada também ao conceito de IoT (Internet of Things), que vem crescendo nos últimos anos, justamente pela sua portabilidade e onipresença.

Outro fato curioso é a presença de Java no sistema operacional Android, que é o mais utilizado nos celulares ao redor do mundo. Isso demonstra a popularidade da linguagem e a confiança que os desenvolvedores têm em Java.

Um aspecto que me chamou a atenção foram os exemplos colocados pelo grupo pois existem padrões e funcionalidades que eu não havia utilizado anteriormente. Como no anexo 7, o uso do setOnClickListener:

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_jogo_adivinha);
        chutar = findViewById(R.id.botao_chutar);

        chutar.setOnClickListener(new View.
           OnClickListener() {
            public void onClick(View v) {

            }
        }
}
```

Listing 12: Exemplo do uso do setOnClickListener

No geral, o trabalho me ajudou a perceber características que eu já usava em Java, mas não sabia a definição e o conceito corretamente. Com esse conhecimento é mais fácil identificar a razão de diversos aspectos implementados na linguagem e como o uso correto desses conceitos melhora a qualidade do código.

Com um aprofundamento em Java o reconhecimento de padrões, estruturas e paradigmas se torna mais intuitivo e as possibilidades de trabalhar a linguagem se tornam mais amplas.O trabalho fez com que eu conhecesse a história, paradigmas, características marcantes e até mesmo diferenças com outras linguagens, assim foi de grande acréscimo para meu desenvolvimento enquanto pessoa que programa em Java.