



Hello, World!

Capítulo 1: O Chamado para Kaer Morhen

JAVA
QUANDO O MEDALHÃO
DETECTA OS SINAIS

`int x = 42;
if (x > 0)`

Introdução ao Mundo da Programação Java

"O caminho é longo e perigoso. Poucos sobreviverão... Mas aqueles que permanecerem serão forjados em algo extraordinário."

— Vesemir, Mestre de Kaer Morhen

1.1 A Conjunção das Esferas Digitais

O que é programação e por que aprender Java

No mundo de The Witcher, a Conjunção das Esferas foi um evento cataclísmico que alterou para sempre a realidade, fundindo mundos distintos e trazendo criaturas místicas, magia e novas possibilidades para um universo anteriormente mundano. De forma semelhante, a programação representa uma conjunção entre o mundo humano e o universo digital, onde nossos pensamentos lógicos e intenções podem ganhar vida através de linguagens que os computadores compreendem.

Programação é, em sua essência, a arte de instruir máquinas. É o processo meticuloso de criar sequências de comandos — algoritmos — que direcionam computadores a executarem tarefas específicas. Assim como um bruxo domina sinais mágicos para manipular elementos da natureza, um programador domina linguagens de programação para moldar o comportamento de máquinas. Esta comunicação não acontece em linguagem humana convencional, mas através de códigos estruturados que seguem regras rigorosas e precisas.

Quando aprendemos a programar, estamos essencialmente adquirindo a capacidade de transformar ideias abstratas em soluções concretas. É como aprender os gestos e encantamentos necessários para lançar um sinal mágico — não basta conhecer o conceito, é preciso dominar sua execução precisa.



Por que Java se destaca nesta jornada?

Imagine Java como a Escola do Lobo entre as linguagens de programação — respeitada, versátil e com tradição comprovada. Criada para sobreviver em ambientes hostis e adaptável a diferentes territórios, Java carrega características que a tornam uma escolha poderosa para iniciantes e veteranos:

1.Portabilidade Interdimensional: Assim como um bruxo pode atravessar diferentes reinos e continuar usando suas habilidades, o código Java funciona em qualquer dispositivo que possua a JVM (Java Virtual Machine). Esta característica, resumida no mantra "Write Once, Run Anywhere" (Escreva Uma Vez, Execute Em Qualquer Lugar), é como um portal mágico que permite que seu código viaje entre sistemas operacionais sem alterações.

2.Linguagem de Alto Nível com Raízes Profundas: Java oferece abstrações que facilitam o desenvolvimento (alto nível), mas mantém conexão com recursos fundamentais do sistema. É como um bruxo que utiliza poções elaboradas (abstrações), mas compreende perfeitamente as

propriedades de cada ingrediente (fundamentação).

- 2. Comunidade Ancestral e Vasta:** Poucos sabem, mas Java é como uma antiga ordem de bruxos com conhecimentos acumulados ao longo de décadas. Sua comunidade extensa criou bibliotecas para praticamente qualquer problema imaginável — é como ter acesso a um grimório infinito de feitiços já testados e aprimorados.
- 3. Demanda Constante por Caçadores:** No universo tecnológico, desenvolvedores Java são como bruxos especializados — sempre requisitados para enfrentar os desafios mais complexos. De sistemas bancários a aplicativos Android, a versatilidade de Java garante que seus praticantes raramente fiquem sem contratos.

Conselhos do Mestre Vesemir:

"Não se engane, jovem aprendiz. O caminho do Java é desafiador no início. Sua sintaxe pode parecer mais verbosa que outras escolas de magia, como Python. Mas esta disciplina rigorosa forja desenvolvedores resilientes. Quando dominar Java, outras linguagens parecerão meros dialetos mais simples da mesma magia fundamental."

Para entender verdadeiramente o poder de Java, precisamos conhecer sua origem e como ela evoluiu através das eras digitais.



História e evolução da linguagem Java

As Origens Místicas: O Projeto Green

Assim como o mundo de The Witcher tem suas lendas sobre a origem dos primeiros bruxos, a história de Java começa em 1991, quando James Gosling, Mike Sheridan e Patrick Naughton, engenheiros da Sun Microsystems, iniciaram uma busca ambiciosa. Este trio pode ser visto como os primeiros mestres alquimistas que buscavam criar algo revolucionário.

O Projeto Green, como foi chamado, nasceu da visão de que dispositivos eletrônicos diversos precisariam se comunicar entre si no futuro — uma previsão que antecipava nossa atual era de dispositivos conectados. Esta visão era revolucionária para a época, comparável à criação das primeiras poções mutagênicas que deram origem aos bruxos.

A linguagem inicialmente foi batizada de "Oak" (Carvalho), em homenagem a uma árvore que Gosling podia ver da janela de seu escritório — uma conexão com a natureza que lembra as raízes profundas que os bruxos mantêm com o mundo natural, apesar de sua transformação. A equipe desenvolveu um dispositivo chamado *7 (StarSeven), que poderia ser considerado o primeiro "medalhão" Java — um controle remoto com tela sensível ao toque e interface gráfica, muito à frente de seu tempo.

A Primeira Mutação: De Oak para Java

No universo de The Witcher, os primeiros bruxos surgiram após experimentos e mutações. Da mesma forma, a linguagem Oak passou por transformações significativas quando a equipe percebeu que o mercado de dispositivos eletrônicos não estava pronto para sua visão. Em 1993-1994, com a ascensão da World Wide Web, uma nova oportunidade surgiu.

A equipe redirecionou seus esforços para a internet emergente, onde a portabilidade de Oak seria uma vantagem inestimável. Por questões legais relacionadas a marcas registradas, Oak foi renomeada para "Java", inspirada pelo café favorito da equipe — assim como bruxos recebem novos nomes ao completarem sua transformação.

Em 23 de maio de 1995, durante a conferência SunWorld, Java foi apresentada ao mundo com uma demonstração impressionante: applets rodando no navegador Netscape Navigator. Era como a primeira demonstração pública dos sinais mágicos dos bruxos — algo que capturou a imaginação e mostrou possibilidades que poucos haviam considerado.

As Eras do Java


| Era | Marcos Principais | Analogia ao Mundo Witcher |
|---------------------------------|---|---|
| Era do Surgimento (1991-1995) | Projeto Green, Oak, Java 1.0 | Conjunção das Esferas - Mundos Convergem |
| Era das Fundações (1996-2006) | Java 1.1 até Java 6, Expansão Enterprise | Fundação das Escolas de Bruxos - Estruturação |
| Era da Modernização (2007-2017) | Aquisição pela Oracle, Java 7 e 8, Lambda | Expansão dos Reinos - Novas Fronteiras |
| Era da Revolução (2018-2025) | Java 9-21, Módulos, Records, Loom | Era da Caça Avançada - Técnicas Refinadas |

A Era das Fundações (1996-2006)

Após seu lançamento oficial, Java iniciou sua expansão como as escolas de bruxos se estabeleceram pelo Continente. A versão Java 1.0 já continha muitos elementos fundamentais que permanecem até hoje, assim como os princípios básicos do treinamento de bruxos.

O período entre 1996 e 2006 viu o crescimento explosivo de Java no desenvolvimento empresarial. Tecnologias como Enterprise JavaBeans (EJB), Servlets, e Java 2 Enterprise Edition (J2EE) estabeleceram Java como a escolha preferida para aplicações corporativas robustas. Este período pode ser comparado à época em que as diferentes escolas de bruxos

(Lobo, Grifo, Gato, etc.) desenvolveram suas técnicas especializadas e se estabeleceram como forças respeitadas.

 **Você sabia?** A versão Java 1.4, lançada em 2002, introduziu a palavra-chave `assert`, que pode ser comparada a um bruxo lançando um feitiço de detecção para verificar se uma condição é verdadeira antes de prosseguir.

A Era das Fundações (1996-2006)

Em 2009, a Sun Microsystems foi adquirida pela Oracle Corporation, marcando o início de uma nova era para Java. Esta transição pode ser comparada ao momento em que algumas escolas de bruxos passaram por reorganizações após grandes batalhas ou mudanças políticas.

As versões Java 7 (2011) e Java 8 (2014) trouxeram inovações significativas. Java 8, em particular, revolucionou a linguagem com a introdução de expressões `lambda` e `streams`, permitindo programação funcional em Java. Esta evolução é comparável a um bruxo veterano descobrindo uma nova técnica de meditação que potencializa todos os seus sinais mágicos existentes.

A Era da Revolução (2018-2025)

A partir de 2018, com o Java 9, a linguagem adotou um ciclo de lançamento acelerado, com novas versões a cada seis meses. Este período trouxe inovações dramáticas:

- **Sistema de Módulos (Java 9):** Como escolas de bruxos organizando seus conhecimentos em grimórios especializados
- **Records (Java 16):** Simplificação para representação de dados imutáveis, como fórmulas de poções aperfeiçoadas
- **Pattern Matching (Java 16+):** Nova forma de estruturar código, comparável a técnicas avançadas de combate
- **Virtual Threads (Projeto Loom, Java 21):** Revolucionou o tratamento de concorrência, como um bruxo aprendendo a dividir sua consciência para controlar múltiplos sinais simultaneamente

Em maio de 2025, Java completou 30 anos de existência, consolidando-se como uma das linguagens mais influentes e duradouras na história da computação. Assim como os bruxos centenários que testemunharam épocas de transformação no Continente, Java adaptou-se e evoluiu, mantendo sua relevância em um mundo tecnológico em constante mudança.



JAVA

Popularidade → %



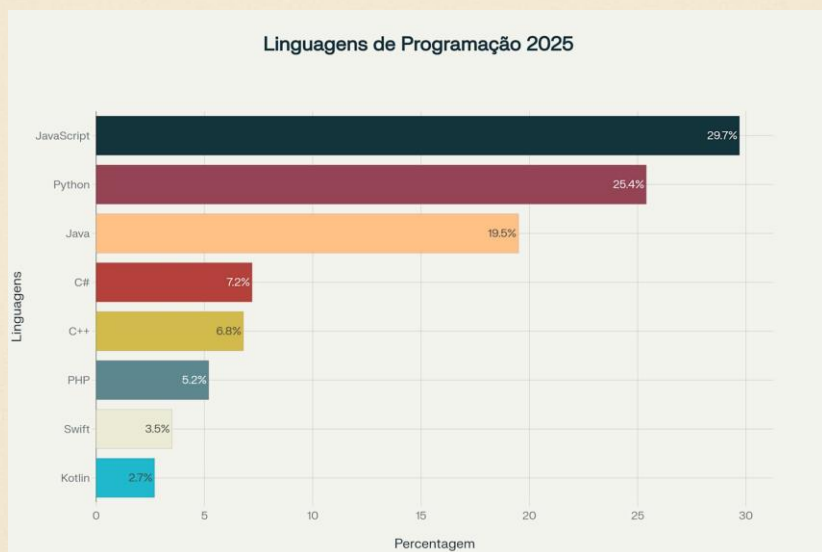
+ ; = / (+ / \ ;
+ % () } + % + /

Java no mercado atual de tecnologia

Em 2025, Java mantém-se como uma das linguagens mais utilizadas globalmente, graças à sua adaptabilidade, robustez e comunidade ativa. Assim como os bruxos continuam relevantes em um mundo que constantemente tenta descartá-los como relíquias do passado, Java demonstra resiliência impressionante diante de linguagens mais jovens.

Relevância e Permanência no Mercado

O gráfico a seguir ilustra a posição de Java em relação a outras linguagens populares em 2025:



Apesar da ascensão de Python e JavaScript, Java mantém uma participação impressionante de 19.5% no mercado de desenvolvimento. Esta resiliência pode ser atribuída a vários fatores:

1. **Base instalada massiva:** Sistemas críticos em bancos, governos e corporações foram construídos em Java há décadas e continuam em operação, assim como antigas fortalezas que ainda servem seu propósito original.
2. **Ecossistema expansivo:** Frameworks como Spring, Hibernate e Apache Kafka formam um arsenal comparável à coleção de armas prateadas, poções e decoctions de um bruxo experiente.
3. **Plataforma Android:** Java permanece fundamental para o desenvolvimento Android, o sistema operacional móvel mais utilizado do mundo. É como um bruxo que se especializa em um tipo específico de monstro que, coincidentemente, é o mais comum no reino.
4. **Evolução constante:** A linguagem continua se adaptando às novas necessidades, como um bruxo que constantemente aprimora suas técnicas para enfrentar novas ameaças.

Áreas de Aplicação em 2025

Java continua sendo a espinha dorsal de diversos setores cruciais:

- **Sistemas financeiros e bancários:** A segurança e confiabilidade de Java são essenciais para sistemas que processam transações financeiras. Como bruxos contratados para proteger tesouros valiosos, Java guarda os ativos mais preciosos da era digital.
- **Desenvolvimento empresarial:** Aplicações corporativas de larga escala continuam sendo domínio do Java, especialmente com o ecossistema Spring. É como uma guilda de bruxos que se especializa em ameaças urbanas complexas.
- **Big Data e processamento analítico:** Tecnologias como Hadoop e Spark, escritas em Java, processam volumes enormes de dados. Compare isso a um bruxo usando seus sentidos ampliados para detectar padrões imperceptíveis aos humanos comuns.
- **Internet das Coisas (IoT):** Versões otimizadas de Java são utilizadas em dispositivos embarcados e soluções IoT. É como um bruxo adaptando suas técnicas para trabalhar em escalas menores, mas não menos importantes.

Perspectivas Profissionais e Remuneração

Desenvolvedores Java continuam sendo altamente valorizados no mercado de trabalho. No Brasil de 2025, os salários para profissionais Java são comparáveis ao pagamento que bruxos recebem por contratos de diferentes níveis de complexidade:

| Nível | Salário Médio (R\$) | Analogia Witcher |
|--------------|---------------------|---|
| Júnior | R\$ 4.800/mês | Aprendiz de Bruxo - Primeiras missões |
| Pleno | R\$ 7.200/mês | Bruxo Formado - Caçadas regulares |
| Sênior | R\$ 12.500/mês | Bruxo Experiente - Domínio das técnicas |
| Especialista | R\$ 18.000+/mês | Mestre Bruxo - Lendas do ofício |

Conselhos do Mestre Vesemir:

"O verdadeiro valor de um bruxo não está apenas em conhecer os movimentos de sua espada, mas em saber quando e onde aplicá-los. Da mesma forma, um desenvolvedor Java não é valioso apenas por conhecer a sintaxe, mas por entender como utilizá-la para resolver problemas reais. Busque sempre compreender o 'porquê' por trás do 'como'."

O Futuro de Java

Assim como os bruxos que se adaptam às mudanças do mundo, Java continua evoluindo. O roadmap atual inclui melhorias significativas em áreas como:

- **Desempenho:** Project Leyden visa reduzir o tempo de inicialização e o consumo de memória
- **Concorrência:** Aprimoramentos nas Virtual Threads e APIs para programação concorrente
- **Sintaxe:** Simplificações para reduzir a verbosidade tradicional do Java
- **Interoperabilidade:** Melhor integração com linguagens nativas e tecnologias modernas

A Oracle, atual guardiã do Java (comparável ao Conselho dos Bruxos), mantém um ciclo de lançamento previsível que garante estabilidade enquanto introduz inovações, equilibrando tradição e evolução.

Contrato Prático: Pesquisa de Mutágenos Tecnológicos

Como um jovem aprendiz chegando a Kaer Morhen, sua primeira tarefa é investigar o panorama atual do Java. Complete as seguintes missões:

- Identifique três empresas que utilizam Java em sua região e quais tipos de aplicações desenvolvem
- Pesquise o salário médio atual para desenvolvedores Java júnior em sua cidade
- Encontre um projeto open-source em Java no GitHub e analise sua estrutura básica



Reflexão Final:

A história de Java, assim como a saga dos bruxos, é uma narrativa de adaptação, resiliência e constante evolução. Desde suas origens místicas no Projeto Green até sua posição atual como uma das linguagens mais utilizadas no mundo, Java demonstra que tecnologias bem fundamentadas podem perdurar através das eras, contanto que continuem evoluindo para atender às necessidades de seu tempo.

Ao embarcar na jornada de aprendizado de Java, você não está apenas dominando uma linguagem de programação – está se conectando a uma tradição rica em história e conhecimento prático, preparando-se para enfrentar os desafios do mundo digital com as ferramentas de um verdadeiro mestre da arte.

"Um bom início faz uma boa jornada. Prepare-se adequadamente, e os monstros do código tremerão diante de seu medalhão Java." — Adaptado dos ensinamentos de Vesemir



"Uma fortaleza não é apenas pedra e argamassa, jovem aprendiz. É um ecossistema completo onde cada parte tem seu propósito específico, trabalhando em harmonia para sustentar aqueles que buscam conhecimento."

— Vesemir, Mestre de Kaer Morhen

Capítulo 1.2 As Fundações de Kaer Morhen

Assim como Kaer Morhen não é meramente uma construção isolada nas montanhas, mas sim um complexo sistema integrado de torres, salões, arsenais e bibliotecas, Java transcende a definição simples de "linguagem de programação".

Para compreender verdadeiramente o poder que está sob seu comando como futuro desenvolvedor Java, você deve primeiro entender que está adentrando um reino tecnológico multifacetado, onde cada componente possui uma função específica e interdependente .

O que é Java: linguagem, plataforma e ecossistema

Java é, simultaneamente, três entidades distintas que trabalham em perfeita sintonia, como os três aspectos fundamentais que tornam Kaer Morhen a fortaleza de treinamento mais eficaz do Continente .

Java como Linguagem de Programação

Em seu primeiro aspecto, Java é uma linguagem de programação orientada a objetos, comparável aos códigos secretos e linguagens mágicas que os bruxos dominam para comunicar conceitos complexos . Esta linguagem possui uma sintaxe estruturada e rigorosa, similar aos gestos precisos necessários para lançar um sinal mágico — cada palavra-chave, cada pontuação tem um significado específico que deve ser respeitado .

A linguagem Java foi projetada com cinco princípios fundamentais, conhecidos como os "Cinco Pilares de Oak" (nome original da linguagem), que podem ser comparados aos cinco sinais básicos dos bruxos:

1. **Simplicidade** — Como Aard, direto e eficaz
2. **Orientação a Objetos** — Como Quen, proteção através de estrutura
3. **Robustez** — Como Yrden, armadilhas que previnem erros
4. **Segurança** — Como sinais de proteção avançados
5. **Portabilidade** — Como Axi, adaptação a diferentes ambientes

💡 **Insight Avançado:** A sintaxe de Java foi intencionalmente inspirada em C++, mas simplificada para eliminar características perigosas como ponteiros e gerenciamento manual de memória . É como aprender esgrima depois de dominar combate com machados — você mantém a precisão, mas ganha elegância e segurança.


Java como Plataforma Tecnológica

O segundo aspecto de Java é sua natureza como plataforma independente, uma característica revolucionária comparável à capacidade dos bruxos de se adaptarem a qualquer reino ou território . Esta independência é alcançada através de um conceito engenhoso: ao invés de compilar código diretamente para linguagem de máquina específica, Java cria um "idioma universal" chamado bytecode .

O processo funciona como um sistema de tradução em três etapas:

```
1 Código Java (.java) → Compilação → Bytecode (.class) → JVM → Código Nativo
```

Este bytecode é como uma linguagem franca comercial que diferentes reinos (sistemas operacionais) podem compreender através de seus próprios intérpretes especializados . Um bruxo pode escrever uma poção em Kaer Morhen e ela funcionará igualmente bem em Oxenfurt, Novigrad ou até mesmo em Skellige, desde que cada local tenha um alquimista capaz de interpretar a receita .

 **Armadilha Comum:** Muitos iniciantes confundem "independência de plataforma" com "funciona em qualquer lugar sem configuração". Java é independente de plataforma, mas ainda requer que a JVM apropriada esteja instalada no sistema de destino — é como um bruxo que pode viajar para qualquer reino, mas ainda precisa que cada reino reconheça e aceite sua presença .

Java como Ecossistema Tecnológico

O terceiro e mais abrangente aspecto de Java é seu vasto ecossistema, comparável à rede completa de escolas de bruxos, bibliotecas, arsenais e conhecimentos acumulados ao longo de séculos . Este ecossistema inclui:

Bibliotecas Padrão: Como o grimório oficial dos bruxos, contendo feitiços testados e aprovados para uso geral .

Frameworks: Comparáveis às técnicas especializadas de cada escola de bruxo (Spring como Escola do Lobo, Hibernate como Escola do Grifo) .

Ferramentas de Desenvolvimento: Como os instrumentos especializados que cada bruxo carrega em sua jornada .

Comunidade Global: Milhões de desenvolvedores que compartilham conhecimentos, similar à rede de bruxos espalhados pelo mundo conhecido .

JVM (Java Virtual Machine) -

O coração da fortaleza

No centro majestoso de Kaer Morhen ergue-se o salão principal, onde pulsa o coração da fortaleza — aqui encontramos nossa analogia perfeita para a Java Virtual Machine (JVM) . A JVM é literalmente o núcleo vital que dá vida a todo código Java, transformando instruções abstratas em ações concretas, como um coração que bombeia sangue vital através de todo o organismo .

Arquitetura Interna da JVM

A arquitetura da JVM é tão complexa e fascinante quanto os sistemas internos de Kaer Morhen, com diferentes áreas especializadas trabalhando em harmonia perfeita :

Class Loader (Porteiro da Fortaleza): O primeiro guardião que recebe visitantes (classes Java) e determina onde cada um deve ser acomodado na fortaleza . Ele verifica credenciais (bytecode), carrega recursos necessários e organiza tudo de forma ordenada .


Memory Areas (Compartimentos da Fortaleza):

- **Heap:** O grande salão onde os objetos vivem e interagem, organizado como um banquete perpétuo onde cada objeto tem seu lugar

- **Stack:** As torres individuais onde cada método executa suas tarefas, como aposentos privados para trabalho concentrado
- **Method Area:** A biblioteca principal onde ficam armazenadas as definições de classes e métodos, como os tomos de conhecimento ancestral

Execution Engine (O Verdadeiro Mestre da Magia): O componente mais fascinante da JVM, responsável por interpretar e executar o bytecode . Este "motor de execução" possui duas capacidades extraordinárias:

1. **Interpretação Direta:** Lê e executa bytecode linha por linha, como um bruxo iniciante que segue cuidadosamente uma receita de poção
2. **Just-In-Time Compilation (JIT):** A verdadeira magia acontece aqui — quando detecta que certas seções de código são executadas repetidamente, o JIT as transforma em código nativo otimizado

 **Detalhe Técnico Aprofundado:** O JIT Compiler é como um bruxo experiente que, após repetir um sinal várias vezes, desenvolve uma forma mais eficiente de executá-lo . O compilador monitora "hot spots" (pontos quentes) no código — seções executadas frequentemente — e as compila para código de máquina nativo, resultando em performance que pode até superar código compilado tradicionalmente .

Garbage Collector (O Zelador Mágico): Um dos aspectos mais impressionantes da JVM é sua capacidade de limpeza automática da memória . Como um zelador mágico que trabalha invisivelmente, o Garbage Collector identifica objetos que não são mais utilizados e libera automaticamente a memória que ocupavam, evitando vazamentos e mantendo a fortaleza sempre organizada .

Tipos de Garbage Collectors em 2025

A JVM moderna oferece diferentes estratégias de coleta de lixo, cada uma adequada para diferentes tipos de "missões" :

| Garbage Collector | Analogia Witcher | Uso Recomendado |
|-------------------|---|--|
| Serial GC | Zelador solitário e metuculoso | Aplicações pequenas, dispositivos com recursos limitados |
| Parallel GC | Equipe de zeladores trabalhando juntos | Aplicações de throughput alto |
| G1 GC | Zelador inteligente que prioriza áreas mais sujas | Aplicações de latência baixa (padrão desde Java 9) |
| ZGC | Zelador quase instantâneo e imperceptível | Aplicações críticas de ultra-baixa latência |

Conselhos do Mestre Vesemir:

"A JVM é como o coração de um bruxo veterano — quanto mais experiente, mais eficiente se torna. O JIT Compiler observa seus padrões de código e os otimiza automaticamente. Por isso, aplicações Java frequentemente ficam mais rápidas quanto mais tempo ficam em execução. Não se surpreenda se seu programa demonstrar melhor performance após alguns minutos de funcionamento."

JDK (Java Development Kit) - As ferramentas do bruxo

Se a JVM é o coração de Kaer Morhen, então o Java Development Kit (JDK) representa o arsenal completo e a biblioteca de conhecimentos que todo bruxo precisa para sua jornada . O JDK é muito mais que um simples conjunto de ferramentas — é um kit completo de sobrevivência para desenvolvedores Java, comparável ao equipamento completo que um bruxo carrega ao partir para uma missão perigosa .

Arquitetura Interna da JVM

O Compilador Java (javac) - A Forja Mágica: O compilador é a ferramenta fundamental que transforma seu código-fonte Java em bytecode, processo comparável à forja onde um ferreiro bruxo transforma metal bruto em uma lâmina letal . Esta transformação não é meramente mecânica — o compilador também atua como um mentor vigilante, identificando erros e inconsistências antes que possam causar problemas .

```
// Comando básico de compilação  
javac MinhaClasse.java // Transforma .java em .class
```


Java Runtime Environment (JRE) Completo: O JDK inclui uma versão completa do JRE, garantindo que você possa não apenas criar programas, mas também executá-los imediatamente . É como ter não apenas as ferramentas para forjar uma espada, mas também a capacidade de testá-la em combate .

Documentação e APIs (javadoc) - O Grimório do Conhecimento: Uma das ferramentas mais valiosas do JDK é o gerador de documentação, que transforma comentários especiais em seu código em documentação profissional . É como ter um escriba mágico que automaticamente registra suas descobertas e técnicas para referência futura.

Depurador Java (jdb) - O Olho de Águia: Ferramenta essencial para investigar problemas em seu código, permitindo que você execute programas passo a passo e examine o estado interno . É comparável à capacidade aprimorada de um bruxo de perceber detalhes imperceptíveis ao olho comum.

Ferramentas Avançadas do Arsenal:

- **jar:** Empacota aplicações em arquivos transportáveis
- **javap:** Desmonta bytecode para análise detalhada
- **jstack:** Analisa threads em execução
- **jstat:** Monitora performance da JVM

 **Você sabia?** O JDK moderno (versão 21) inclui mais de 200 ferramentas especializadas, cada uma projetada para uma tarefa específica no desenvolvimento Java . É como ter acesso ao arsenal completo de todas as escolas de bruxos simultaneamente.

Versões e Distribuições do JDK

Em 2025, o ecossistema Java oferece múltiplas distribuições do JDK, cada uma adaptada para diferentes necessidades, como diferentes tradições de escolas de bruxos :

Oracle JDK: A versão "oficial" mantida pela Oracle, com suporte comercial de longo prazo .

OpenJDK: A versão open-source e gratuita, base para todas as outras distribuições .

Distribuições Especializadas:

- **Amazon Corretto:** Otimizada para nuvem AWS
- **Eclipse Adoptium:** Distribuição comunitária amplamente utilizada
- **Azul Zulu:** Focada em performance enterprise
- **GraalVM:** Versão revolucionária com capacidades nativas

JDK (Java Development Kit) - As ferramentas do bruxo

O **Java Runtime Environment (JRE)** é como o equipamento essencial que um bruxo veterano leva em missões específicas — contém apenas o necessário para executar tarefas sem o peso adicional de ferramentas de criação . Enquanto o JDK é o arsenal completo de Kaer Morhen, o JRE é o kit cuidadosamente selecionado para operações em campo .

Composição e Função do JRE

O JRE combina três elementos fundamentais em um pacote coeso e otimizado :

1. **JVM Otimizada:** Uma versão da Java Virtual Machine configurada especificamente para execução, sem ferramentas de desenvolvimento . É como um bruxo que partiu apenas com suas habilidades essenciais, deixando para trás livros de estudo e equipamentos de treinamento .
2. **Bibliotecas de Classes Java:** O conjunto completo de bibliotecas padrão (Java Standard Library) que fornece funcionalidades prontas para uso . Esta biblioteca é como uma coleção de poções e ingredientes essenciais que permitem ao bruxo enfrentar a maioria das situações sem precisar improvisar .

4. **Arquivos de Suporte:** Configurações, fonts, propriedades e outros recursos necessários para execução adequada . São como os itens de apoio que um bruxo experiente sempre carrega — não são armas ou poções, mas essenciais para o funcionamento adequado do equipamento principal .

Casos de Uso do JRE

Servidores de Produção: Em ambientes de produção, frequentemente se utiliza apenas o JRE para executar aplicações Java já compiladas, reduzindo a superfície de ataque e consumo de recursos . É como uma guarnição de soldados especializados apenas em combate, sem necessidade de equipamentos de treinamento .

Aplicações Desktop: Usuários finais que executam programas Java em seus computadores geralmente precisam apenas do JRE instalado . É como cidadãos que se beneficiam da proteção de bruxos sem precisar conhecer os segredos do treinamento .

Contêineres e Microserviços: No desenvolvimento moderno, imagens Docker frequentemente incluem apenas o JRE para otimizar tamanho e performance . É como pelotões de reconhecimento que levam apenas equipamento essencial para missões específicas .

Diferença Prática entre JDK, JRE e JVM

Para cristalizar o entendimento, imagine que você está preparando diferentes expedições a partir de Kaer Morhen :

| Componente | Analogia | Quando Usar |
|------------|---------------------------------------|--|
| JVM | O coração/alma do bruxo | Nunca usado isoladamente - sempre parte do JRE |
| JRE | Bruxo veterano em missão | Executar aplicações Java já criadas |
| JDK | Bruxo + arsenal completo + biblioteca | Desenvolver, compilar e executar aplicações Java |

Relação de Inclusão:

```
JDK = JRE + Ferramentas de Desenvolvimento
JRE = JVM + Bibliotecas de Classes + Arquivos de Suporte
JVM = Motor de execução isolado (nunca usado sozinho)
```

⚠ Armadilha para Iniciantes: Uma confusão comum é tentar instalar apenas a JVM . Na prática, você sempre instalará pelo menos o JRE (que inclui a JVM) ou o JDK completo (que inclui JRE + ferramentas). É como tentar partir em uma missão levando apenas seu coração — você precisa do corpo completo (JRE) e, para criar novas técnicas, do arsenal completo (JDK) .

Conselhos do Mestre Vesemir:

"Assim como um bruxo experiente conhece cada peça de seu equipamento, você deve compreender profundamente estes três pilares do Java. A JVM é sua força vital, o JRE é seu equipamento de campo, e o JDK é seu arsenal completo. Dominar como eles trabalham juntos é o primeiro passo para se tornar um verdadeiro mestre do Java. Lembre-se: você pode possuir as melhores ferramentas do mundo, mas sem compreender como utilizá-las em harmonia, elas não passam de metal frio."

Com esta compreensão sólida das fundações de Kaer Morhen, você está preparado para começar sua verdadeira jornada como desenvolvedor Java. No próximo capítulo, colocaremos a mão na massa e prepararemos seu ambiente de desenvolvimento, transformando conhecimento teórico em habilidade prática.



"Todo bruxo precisa de suas ferramentas, jovem aprendiz. Uma espada sem fio não corta, uma poção sem ingredientes não cura, e um desenvolvedor sem ambiente configurado não programa." — Vesemir, preparando um novo aprendiz

1.3 Preparando-se para a Jornada

Chegou o momento mais emocionante de sua jornada — transformar conhecimento teórico em habilidade prática. Assim como um jovem aprendiz deve preparar cuidadosamente seu equipamento antes de enfrentar sua primeira caçada, você precisa configurar adequadamente seu ambiente de desenvolvimento Java antes de escrever sua primeira linha de código.

Esta preparação não é apenas uma formalidade técnica, mas um ritual de iniciação que marca sua transição de espectador para praticante ativo da arte da programação Java. Cada ferramenta instalada, cada configuração ajustada, representa um passo crucial em direção ao domínio completo desta linguagem poderosa.

Instalação e configuração do ambiente de desenvolvimento

Escolhendo Suas Armas Digitais

Antes de partir para Kaer Morhen, todo aspirante a bruxo deve escolher cuidadosamente suas armas e ferramentas. No mundo Java, suas "armas" principais são o **JDK (Java Development Kit)** e um **IDE (Integrated Development Environment)** adequado. Esta escolha inicial determinará sua eficiência e produtividade ao longo de toda a jornada.

Download e Instalação do JDK

Para 2025, recomendamos o **Java 21 LTS (Long Term Support)**, uma versão estável e moderna que será suportada por anos. É como escolher uma espada forjada pelos melhores ferreiros — durável, confiável e com recursos avançados.

No Windows:

1. Acesse o site oficial da Oracle (oracle.com/java) ou Eclipse Adoptium (adoptium.net)
2. Baixe o instalador do JDK 21 para Windows x64
3. Execute o instalador como administrador
4. Siga as instruções, mantendo o diretório padrão:

```
C:\Program Files\Java\jdk-21
```

No macOS:

1. Baixe o arquivo *.dmg* do JDK 21
2. Monte a imagem e execute o instalador
3. O JDK será instalado em
`/Library/Java/JavaVirtualMachines/`

No Linux (Ubuntu/Debian):

```
# Atualizar repositórios
sudo apt update

# Instalar JDK 21
sudo apt install openjdk-21-jdk

# Verificar instalação
java --version
javac --version
```


Configuração das Variáveis de Ambiente

As variáveis de ambiente são como runas de orientação que guiam o sistema para encontrar suas ferramentas Java. Sem elas, é como tentar navegar pelas Montanhas Azuis sem bússola.

JAVA_HOME - O Caminho da Fortaleza:

Windows:

1. Clique com botão direito em "Este Computador" → Propriedades
2. Configurações Avançadas do Sistema → Variáveis de Ambiente
3. Novo (Variáveis do Sistema):
 - Nome: `JAVA_HOME`
 - Valor: `C:\Program Files\Java\jdk-21` (ou seu caminho de instalação)


Linux/macOS:

```
# Adicionar ao arquivo ~/.bashrc ou ~/.zshrc
export JAVA_HOME=/usr/lib/jvm/java-21-openjdk-amd64
export PATH=$PATH:$JAVA_HOME/bin

# Recarregar configurações
source ~/.bashrc
```

PATH - As Rotas de Acesso:

Adicione `%JAVA_HOME%\bin` (Windows) ou `$JAVA_HOME/bin` (Linux/macOS) à variável PATH do sistema.

 **Verificação da Configuração:** Abra um novo terminal e execute:

```
java --version
javac --version
echo $JAVA_HOME # Linux/macOS
echo %JAVA_HOME% # Windows
```

Se todos os comandos retornarem informações sobre Java 21, sua configuração está perfeita!

Escolhendo e Configurando seu IDE

Um IDE é como a bancada de trabalho personalizada de um bruxo experiente — um local onde todas as ferramentas estão organizadas e acessíveis. Para iniciantes em Java, recomendamos três opções principais:

IntelliJ IDEA Community (Recomendação Principal):

- Gratuito, poderoso e intuitivo
- Autocomplete inteligente como um grímório que se escreve sozinho
- Debugging visual comparável aos sentidos aprimorados de um bruxo
- Download: jetbrains.com/idea/download

Eclipse IDE:

- Tradicional, estável e extensível
- Vasta comunidade e plugins disponíveis
- Interface mais tradicional, ideal para quem prefere controle total
- Download: eclipse.org/downloads

Visual Studio Code com extensões Java:

- Leve e versátil
- Excelente para quem gosta de personalização
- Requer configuração adicional, mas muito flexível
- Download: code.visualstudio.com

Conselhos do Mestre Vesemir:

"Assim como cada bruxo desenvolve preferência por certas armas, você desenvolverá afinidade com determinado IDE. Comece com IntelliJ IDEA Community — é como aprender com a Escola do Lobo: métodos testados e aprovados. Quando dominar o básico, experimente outras ferramentas."

Primeira linha de código: "Olá, Mundo!"

Chegou o momento mais emocionante — escrever seu primeiro feitiço em linguagem Java! Tradicionalmente, todo programador iniciante escreve um programa "Hello, World!" como seu primeiro ritual de iniciação. Nós adaptaremos esta tradição ao nosso universo temático.

Criando Seu Primeiro Sinal Mágico

Abra seu IDE escolhido e crie um novo projeto Java. Nomeie-o "*MeuPrimeiroSinal*" — este será seu primeiro encantamento funcional no reino do Java.

```
1 /**
2  * MeuPrimeiroSinal.java
3  * O primeiro feitiço de um jovem aprendiz em Kaer Morhen
4  *
5  * Este programa demonstra a estrutura básica de uma aplicação Java
6  * e exibe uma mensagem de boas-vindas ao mundo da programação.
7  */
8 public class MeuPrimeiroSinal {
9
10     /**
11      * Método principal - O ponto de partida de toda aplicação Java
12      * É como o gesto inicial que ativa um sinal mágico
13      *
14      * @param args argumentos da linha de comando (não utilizados neste exemplo)
15      */
16     public static void main(String[] args) {
17         // Primeira linha mágica - exibe uma mensagem no console
18         System.out.println("🔥 O fogo de Igñi está aceso!");
19
20         // Múltiplas invocações para demonstrar sequência
21         System.out.println("♥ O escudo de Quen está ativo!");
22         System.out.println("⚡ O choque de Aard ecoa pelas montanhas!");
23         System.out.println("★ Axii sussurra nos ventos gelados!");
24         System.out.println("○ Yrden forma círculos no chão!");
25
26         // Mensagem de boas-vindas personalizada
27         System.out.println(); // Linha em branco para separação
28         System.out.println("=".repeat(50));
29         System.out.println("🏰 Bem-vindo à Kaer Morhen, jovem aprendiz!");
30         System.out.println("🗡 Sua jornada como desenvolvedor Java começou!");
31         System.out.println("=".repeat(50));
32     }
33 }
```

Anatomia do Primeiro Feitiço


Cada linha deste código tem um propósito específico, como cada gesto em um sinal mágico:

Comentários Explicativos (`/** */` e `//`): Como anotações em um grímório, explicam o propósito e funcionamento do código para outros desenvolvedores (e para você mesmo no futuro).

Declaração da Classe (`public class MeuPrimeiroSinal`): Define o nome do seu "feitiço" — deve corresponder exatamente ao nome do arquivo `.java`.

Método Main (`public static void main`): O coração pulsante do programa — onde a magia acontece. Todo programa Java executável precisa deste método.

***System.out.println()*:** O comando que materializa suas palavras na tela, como invocar um sinal que projeta mensagens visíveis.

 **Insight Avançado:** O método `String.repeat(50)` utilizado para criar a linha de separação é um recurso moderno do Java (introduzido na versão 11). É como descobrir uma versão aprimorada de um sinal antigo — mais eficiente e elegante.

Compilação e execução de programas Java

A compilação em Java é um processo de duas etapas, similar ao preparo e ativação de uma poção mágica. Primeiro você prepara os ingredientes (compilação), depois você ativa seus efeitos (execução).

O Ritual da Compilação

Via Linha de Comando (Método Tradicional):

1. Navegue até o diretório do seu arquivo: Use *cd* para chegar à pasta contendo `MeuPrimeiroSinal.java`
2. Compile o código-fonte:

```
javac MeuPrimeiroSinal.java
```

3. Verifique a criação do bytecode: Um arquivo `MeuPrimeiroSinal.class` deve aparecer no mesmo diretório
4. Execute o programa:

```
java MeuPrimeiroSinal
```

⚠ Armadilha Comum:

Note que na execução usamos `java MeuPrimeiroSinal` (sem extensão), não `java MeuPrimeiroSinal.class`. É como chamar um bruxo pelo nome, não pela descrição completa de suas roupas.

Via IDE (Método Recomendado para Iniciantes):

No IntelliJ IDEA ou Eclipse:

1. Clique com botão direito no arquivo `.java`
2. Selecione "*Run 'MeuPrimeiroSinal.main()'*"
3. O IDE automaticamente compila e executa o programa

Compreendendo o Processo

```
Código Java (.java) → [javac] → Bytecode (.class) → [JVM] → Resultado
```

Etapla 1 - Compilação: O compilador javac verifica seu código Java em busca de erros de sintaxe e, se tudo estiver correto, gera o bytecode — uma linguagem intermediária que a JVM compreende.

Etapla 2 - Execução: A JVM lê o bytecode e o traduz para instruções específicas do sistema operacional onde está rodando.

Esta separação é genial — é como escrever uma receita de poção em linguagem universal que qualquer alquimista pode seguir, independentemente do reino onde esteja.



Detalhe Técnico: O bytecode gerado é verdadeiramente independente de plataforma. Um arquivo *.class* compilado no Windows funcionará perfeitamente no Linux ou macOS, desde que ambos tenham a JVM apropriada instalada.

Estrutura básica de um arquivo Java

Todo arquivo Java segue uma estrutura organizacional rigorosa, como as regras arquitetônicas que regem a construção de fortalezas. Compreender esta estrutura é fundamental para escrever código limpo e manutenível.

Anatomia Completa de um Arquivo Java

```
1 // 1. COMENTÁRIOS DE CABEÇALHO (Opcional, mas recomendado)
2 /**
3  * Nome do Arquivo: ExemploCompleto.java
4  * Propósito: Demonstrar estrutura completa de classe Java
5  * Autor: [Seu Nome]
6  * Data: [Data de Criação]
7  * Versão: 1.0
8  */
9
10 // 2. DECLARAÇÃO DE PACKAGE (Opcional para arquivos simples)
11 package com.kaermorhen.exemplos;
12
13 // 3. IMPORTS (Importações de bibliotecas externas)
14 import java.time.LocalDateTime;
15 import java.util.Scanner;
16
17 // 4. COMENTÁRIO DA CLASSE
18 /**
19  * Esta classe demonstra a estrutura completa de um arquivo Java,
20  * incluindo todos os elementos fundamentais que um desenvolvedor
21  * deve conhecer.
22  */
23
24 // 5. DECLARAÇÃO DA CLASSE PRINCIPAL
25 public class ExemploCompleto {
26
27     // 6. ATRIBUTOS/CAMPOS DA CLASSE (Variáveis de instância)
28     private String nomeAprendiz;
29     private int nivelHabilidade;
30
31     // 7. CONSTANTES (Variáveis estáticas finais)
32     public static final String ESCOLA = "Kaer Morhen";
33     public static final int IDADE_MINIMA = 16;
34
35     // 8. CONSTRUTOR(ES)
36     /**
37      * Construtor padrão - inicializa um novo aprendiz
38      */
39     public ExemploCompleto() {
40         this.nomeAprendiz = "Aprendiz Desconhecido";
41         this.nivelHabilidade = 1;
42     }
```



```

43  /**
44   * Construtor com parâmetros - inicializa aprendiz com dados específicos
45   */
46  public ExemploCompleto(String nome, int nivel) {
47      this.nomeAprendiz = nome;
48      this.nivelHabilidade = nivel;
49  }
50
51  // 9. MÉTODO PRINCIPAL (Ponto de entrada da aplicação)
52  /**
53   * Método main - onde a execução do programa começa
54   */
55  public static void main(String[] args) {
56      // Criando instância da classe
57      ExemploCompleto aprendiz = new ExemploCompleto("Geralt", 99);
58
59      // Chamando métodos da instância
60      aprendiz.apresentarse();
61      aprendiz.demonstrarHabilidades();
62
63      // Exemplo de interação com usuário
64      aprendiz.interagirComUsuario();
65  }
66
67  // 10. MÉTODOS DE INSTÂNCIA
68  /**
69   * Método que apresenta o aprendiz
70   */
71  public void apresentarse() {
72      System.out.println("🏠 Escola: " + ESCOLA);
73      System.out.println("👤 Nome: " + nomeAprendiz);
74      System.out.println("★ Nivel: " + nivelHabilidade);
75      System.out.println("📅 Data atual: " + LocalDateTime.now());
76  }
77
78  /**
79   * Demonstra diferentes habilidades baseadas no nível
80   */
81  public void demostrarHabilidades() {
82      System.out.println("\n🔪 Habilidades Disponíveis:");
83
84      if (nivelHabilidade ≥ 1) {
85          System.out.println("✅ Igni (Fogo básico)");
86      }
87      if (nivelHabilidade ≥ 20) {
88          System.out.println("✅ Quen (Escudo protetor)");
89      }
90      if (nivelHabilidade ≥ 40) {
91          System.out.println("✅ Aard (Onda de choque)");
92      }
93      if (nivelHabilidade ≥ 60) {
94          System.out.println("✅ Axi (Controle mental)");
95      }
96      if (nivelHabilidade ≥ 80) {
97          System.out.println("✅ Yrden (Armadilha mágica)");
98      }
99  }
100

```

```

101  /**
102   * Exemplo de interação básica com o usuário
103   */
104  public void interagirComUsuario() {
105      Scanner entrada = new Scanner(System.in);
106
107      System.out.print("\n👉 Qual sinal você gostaria de aprender primeiro? ");
108      String resposta = entrada.nextLine();
109
110      System.out.println("💡 Excelente escolha! " + resposta +
111          " é um sinal poderoso para iniciantes.");
112
113      entrada.close();
114  }
115
116  // 11. MÉTODOS GETTERS E SETTERS (Acessores)
117  public String getNomeAprendiz() {
118      return nomeAprendiz;
119  }
120
121  public void setNomeAprendiz(String nome) {
122      this.nomeAprendiz = nome;
123  }
124
125  public int getNivelHabilidade() {
126      return nivelHabilidade;
127  }
128
129  public void setNivelHabilidade(int nivel) {
130      if (nivel > 1 && nivel <= 100) {
131          this.nivelHabilidade = nivel;
132      }
133  }
134

```

🔍 O código completo incluindo demais implementações e todos os demais detalhes da criação deste e-book pode ser encontrado no meu repositório do GitHub:

[weritonpetreca/ebook-java-gwent](https://github.com/weritonpetreca/ebook-java-gwent)

Elementos Estruturais Explicados

1. Comentários de Cabeçalho

Como anotações em um pergaminho antigo, estabelecem o propósito e autoria do arquivo. Essencial para documentação profissional e rastreamento de versões.

2. Declaração de Package

Como o endereço da fortaleza onde sua classe reside. Organiza classes em hierarquias lógicas, similar aos diferentes salões de Kaer Morhen.

3. Import Statements

Declara quais bibliotecas externas você utilizará — como listar os grimórios que consultará. Permite acesso a funcionalidades de outras classes e pacotes.

4. Comentário da Classe

Explica o propósito geral da classe — como a placa na entrada de cada salão da fortaleza explicando sua função.

5. Declaração da Classe Principal

Define o nome e características principais da sua classe — o nome de seu feitiço. Estabelece visibilidade (*public*) e tipo (*class*).

6. Atributos/Campos da Classe

Variáveis que armazenam o estado da classe — como os ingredientes de uma poção. Representam as características que cada objeto da classe possuirá.

7. Constantes

Variáveis estáticas finais que nunca mudam — como as leis imutáveis da magia. Usadas para valores que devem permanecer constantes em toda a aplicação.

8. Construtores

Métodos especiais que inicializam objetos — como os rituais de criação de poções. Diferentes construtores permitem diferentes formas de criar objetos.

9. Método Principal

O ponto de entrada da aplicação onde tudo começa — como o ritual principal que ativa todos os outros feitiços. Todo programa Java executável deve ter este método.

10. Métodos de Instância

As habilidades que sua classe pode executar — os sinais que um bruxo pode lançar. Definem o comportamento e ações que objetos da classe podem realizar.

11. Métodos Getters e Setters

Acessores que controlam como os atributos privados são lidos e modificados — como guardas que protegem os tesouros da fortaleza, permitindo acesso controlado às informações internas da classe.

Regras Fundamentais da Estrutura

| Elemento | Regra | Analogia Witcher |
|-------------------|--|--|
| Nome da Classe | Deve corresponder ao nome do arquivo | Como um bruxo, tem apenas um nome verdadeiro |
| Método main | Deve ser <code>public static void main(String[] args)</code> | Ritual de ativação padrão para todos os feitiços |
| Chaves {} | Delimitam blocos de código | Círculos mágicos que contêm poder |
| Ponto e vírgula ; | Termina declarações | Palavra final que sela o encantamento |
| Indentação | 4 espaços por nível (convenção) | Organização hierárquica do conhecimento |

Exercício Prático:

Criando seu primeiro "medalhão"

Chegou o momento de forjar seu primeiro medalhão Java — um programa que não apenas funciona, mas demonstra conceitos fundamentais da linguagem. Este medalhão será interativo e personalizado, como o medalhão único de cada bruxo.

Especificações do Medalhão

Seu programa criará um medalhão personalizado que:

1. Solicita informações do usuário (nome, escola preferida, sinal favorito)
2. Processa essas informações usando estruturas condicionais
3. Exibe um medalhão ASCII personalizado
4. Demonstra diferentes tipos de dados e operações básicas

Objetivos do Exercício

Este exercício consolida conceitos aprendidos no Capítulo 1:



Conceitos Técnicos Demonstrados:

- Entrada e saída de dados com `Scanner`
- Diferentes tipos de dados (`String`, `int`)
- Estruturas condicionais (`switch`, `if-else`)
- Métodos estáticos e passagem de parâmetros
- Operações com Strings (`toLowerCase()`, `toUpperCase()`)
- Formatação de data e hora
- Operações matemáticas básicas



Boas Práticas Aplicadas:

- Comentários explicativos em cada método
- Nomes de variáveis descritivos
- Separação de responsabilidades em métodos
- Validação básica de entrada
- Formatação clara da saída

Desafios Opcionais para Aprimoramento:

- Adicione validação para entradas inválidas
- Implemente um sistema de pontuação baseado nas respostas
- Crie mais opções de personalização do medalhão
- Adicione salvamento das informações em arquivo

Execução e Resultados Esperados

Ao executar este programa, você verá:

- Interface amigável com arte ASCII
- Interação personalizada baseada nas suas respostas
- Medalhão único gerado com seus dados
- Previsão de progresso calculada matematicamente
- Resumo técnico da sessão

Conselhos do Mestre Vesemir:

"Este primeiro medalhão marca sua transformação de observador para praticante. Cada linha de código que você escreveu é como uma cicatriz de treinamento — evidência de que você enfrentou o desafio e emergiu mais forte. Guarde este programa como lembrança de onde sua jornada começou."

Código do Medalhão Interativo

```
1  /**
2   * MedalhaoPersonalizado.java
3   *
4   * Primeiro projeto prático - Criação de um medalhão interativo
5   * que representa a identidade única de cada desenvolvedor Java iniciante.
6   *
7   * Este programa demonstra:
8   * - Entrada e saída de dados
9   * - Variáveis e tipos de dados
10  * - Estruturas condicionais
11  * - Operações com strings
12  * - Formatação de saída
13  */
14
15 import java.util.Scanner;
16 import java.time.LocalDateTime;
17 import java.time.format.DateTimeFormatter;
18
19 public class MedalhaoPersonalizado {
20
21     public static void main(String[] args) {
22         // Criando scanner para entrada de dados
23         Scanner entrada = new Scanner(System.in);
24
25         // Exibindo cabeçalho do programa
26         exibirCabeçalho();
27
28         // Coletando informações do usuário
29         System.out.print("👤 Digite seu nome, jovem aprendiz: ");
30         String nomeAprendiz = entrada.nextLine();
31
32         System.out.print("🐾 Escolha sua escola (Lobo/Grifo/Gato/Vibora/Urso): ");
33         String escolaEscolhida = entrada.nextLine();
34
35         System.out.print("⚡ Qual seu sinal favorito (Igni/Quen/Aard/Axii/Yrden): ");
36         String sinalFavorito = entrada.nextLine();
37
38         System.out.print("🕒 Quantas horas você pretende estudar Java por semana? ");
39         int horasEstudo = entrada.nextInt();
40
41         // Processando informações e determinando características
42         String simboloEscola = determinarSimboloEscola(escolaEscolhida);
43         String corSinal = determinarCorSinal(sinalFavorito);
44         String nivelDedicacao = calcularNivelDedicacao(horasEstudo);
45         String previsaoProgresso = calcularPrevisaoProgresso(horasEstudo);
46
47         // Exibindo medalhão personalizado
48         exibirMedalhao(nomeAprendiz, escolaEscolhida, simboloEscola,
49             sinalFavorito, corSinal, nivelDedicacao, previsaoProgresso);
50
51         // Salvando dados da sessão
52         exibirResumoSessao(nomeAprendiz, horasEstudo);
53
54         entrada.close();
55     }
56
57     /**
58     * Exibe o cabeçalho artístico do programa
59     */
60     public static void exibirCabeçalho() {
61         System.out.println("
62         System.out.println("🏰 FORJA DE MEDALHÕES JAVA 🏰");
63         System.out.println("
64         System.out.println("    Kaer Morhen Academy    ");
65         System.out.println();
66         System.out.println("Bem-vindo à Forja Ancestral de Medalhões!");
67         System.out.println("Aqui você criará seu primeiro artefato Java personalizado.");
68         System.out.println();
69     }
```



```

70
71 /**
72  * Determina o símbolo da escola baseado na escolha do usuário
73  */
74 public static String determinarSímboloEscola(String escola) {
75     String escolhaLower = escola.toLowerCase();
76
77     switch (escolhaLower) {
78         case "lobo":
79             return "🐺";
80         case "grifo":
81             return "🐉";
82         case "gato":
83             return "😺";
84         case "viborá":
85             return "🐍";
86         case "urso":
87             return "🐻";
88         default:
89             return "🦊"; // Símbolo genérico para escolas não reconhecidas
90     }
91 }
92
93
94 /**
95  * Determina a cor/elemento do sinal escolhido
96  */
97 public static String determinarCorSinal(String sinal) {
98     String sinalLower = sinal.toLowerCase();
99
100    switch (sinalLower) {
101        case "igni":
102            return "🔥 FOGO";
103        case "quen":
104            return "🛡️ PROTEÇÃO";
105        case "aard":
106            return "⚡ FORÇA";
107        case "axii":
108            return "🧠 MENTE";
109        case "yrden":
110            return "🛡️ ARMADILHA";
111        default:
112            return "🌟 MAGIA";
113    }
114 }
115
116 /**
117  * Calcula o nível de dedicação baseado nas horas de estudo
118  */
119 public static String calcularNívelDedicacao(int horas) {
120     if (horas > 20) {
121         return "LENDÁRIO 🏆";
122     } else if (horas > 15) {
123         return "DEVOTADO ★★ ★";
124     } else if (horas > 10) {
125         return "DEDICADO ★★ ★";
126     } else if (horas > 5) {
127         return "INICIANTE ★";
128     } else {
129         return "CASUAL 🤷";
130     }
131 }
132
133 /**
134  * Calcula uma previsão de progresso baseada no tempo de estudo
135  */
136 public static String calcularPrevisaoProgresso(int horas) {
137     int semanasParaDominio = (int) Math.ceil(200.0 / horas); // 200 horas para domínio básico
138
139     if (semanasParaDominio <= 10) {
140         return "Você dominará Java em " + semanasParaDominio + " semanas! 🚀";
141     } else if (semanasParaDominio <= 20) {
142         return "Domínio esperado em " + semanasParaDominio + " semanas. 🍀";
143     } else {
144         return "Jornada de " + semanasParaDominio + " semanas pela frente. 🏔️";
145     }
146 }

```

[illegible]

weritonpetreca/ebook-java-gwent

Reflexão e Próximos Passos

Parabéns! Você completou seu primeiro programa Java completo. Este medalhão representa muito mais que código funcional — é evidência tangível de que você:

- ✓ Configurou com sucesso seu ambiente de desenvolvimento
- ✓ Compreendeu a estrutura básica de programas Java
- ✓ Aplicou conceitos de entrada, processamento e saída
- ✓ Utilizou diferentes tipos de dados e estruturas de controle
- ✓ Escreveu código limpo e bem documentado

Este capítulo estabeleceu as fundações sólidas sobre as quais construiremos conhecimentos mais avançados. No próximo capítulo, *“A Anatomia do Bruxo”*, exploraremos os fundamentos da linguagem Java — os elementos fundamentais que compõem o código Java.

“Todo grande bruxo começou exatamente onde você está agora — com curiosidade, determinação e seu primeiro medalhão forjado com as próprias mãos. O caminho à frente é longo, mas cada passo que você dá o torna mais poderoso.” — Reflexões finais de Vesemir

Preparação para o Capítulo 2: Nos próximos dias, pratique executando e modificando seu programa medalhão. Experimente alterar valores, adicionar novos recursos, e familiarize-se completamente com o processo de compilação e execução. Sua confiança com essas operações básicas será essencial para os desafios mais complexos que virão.