



FJOO

Fundamentos Java e Orientação a Objetos

[versão em desenvolvimento]

www.algaworks.com

ALGAWORKS SOFTWARES E TREINAMENTOS

FJOO – Fundamentos Java e Orientação a Objetos

3ª Edição

Setembro/2011

www.algaworks.com

+55 (34) 3255-9898

treinamentos@algaworks.com

Av. Monsenhor Eduardo, 983, Sala 06 A, Bairro Bom Jesus

Uberlândia-MG, CEP. 38400-748

Sobre a empresa

A AlgaWorks é uma empresa localizada em Uberlândia/MG, que trabalha para fornecer treinamentos em TI de qualidade e softwares que realmente funcionam.

Estamos no mercado desde 2004 trabalhando com treinamentos e consultorias sobre a plataforma Java.

Nossa missão

Capacitar pessoas em tecnologias e metodologias de desenvolvimento de software e fornecer aplicativos na internet de baixo custo que contribuem para a organização da vida de milhares de usuários, micro e pequenas empresas de todo o mundo.

Nossa visão

Ser reconhecida como uma das principais empresas de treinamentos em TI do Brasil e como a principal exportadora de software web de baixo custo para indivíduos e pequenas empresas do mundo.

Nossos valores

- Honestidade
- Transparência
- Respeito
- Excelência
- Simplicidade

Origem de nosso nome

"As algas produzem energia necessária ao seu próprio metabolismo através da fotossíntese ... As algas azuis foram os primeiros seres vivos a aparecerem na Terra, e acredita-se que tenham tido um papel fundamental na formação do oxigênio da atmosfera."

A palavra em inglês "Works", além de trabalho, pode ser traduzido como algo que funciona, que tem sucesso.

AlgaWorks foi o nome dado para a nossa empresa, e quer dizer que:

- Queremos ser uma empresa independente, que através de nosso próprio esforço conseguimos produzir uma energia suficiente para a nossa sobrevivência;
- Não queremos depender de investidores externos que não agregam conhecimento ao negócio;
- Desejamos criar alianças com empresas, autônomos, estudantes e profissionais e transmitir todo o nosso conhecimento em tecnologia;
- Fornecemos softwares que funcionam e agregam valor ao negócio, sem complicações;

- Tudo isso deve funcionar com sucesso para ajudar nossos clientes, parceiros, fornecedores, colaboradores e acionistas serem mais felizes.

Sobre esta apostila

Esta apostila faz parte do material didático distribuído pela AlgaWorks para ministrar o curso **Fundamentos Java e Orientação a Objetos**.

Apesar da distribuição desta apostila para fins **não-comerciais** ser permitida, recomendamos que você sempre prefira compartilhar o link da página de download (<http://www.algaworks.com/treinamentos/apostilas>) ao invés do arquivo, pois atualizamos nosso material didático constantemente para corrigir erros e incluir melhorias.

Para você realmente aproveitar o conteúdo desta apostila e aprender a plataforma Java, é necessário que você já tenha conhecimento prévio de sistemas operacionais básicos (Windows ou Linux) e lógica de programação.

Se você quiser se especializar ainda mais em Java ou outras tecnologias, sugerimos que faça os cursos presenciais na AlgaWorks, pois, além de usarmos este material didático ou outros com qualidade semelhante, você ficará por dentro das últimas novidades e terá a chance de aprender com a experiência de nossos instrutores.

Licença desta obra

O conteúdo desta apostila está protegido nos termos da licença **Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported** (http://creativecommons.org/licenses/by-nc-nd/3.0/deed.pt_BR) para a **AlgaWorks Softwares, Treinamentos e Serviços Ltda**, CNPJ **10.687.566/0001-97**. Isso significa que você pode compartilhar (copiar, distribuir e transmitir) esta obra para uso não-comercial, desde que sempre atribua os créditos à AlgaWorks. É vedado a criação de materiais derivados, portanto, você não pode alterar, transformar ou criar algo em cima dessa obra.



Para qualquer reutilização ou distribuição, você deve deixar claro a terceiros os termos da licença a que se encontra submetida esta obra.

Qualquer das condições acima pode ser renunciada se você obtiver permissão da AlgaWorks. Para uso comercial, você deverá comprar os direitos de uso limitados a um número específico de alunos. Neste caso, fale com nossa área comercial.

Encontrou erros ou gostaria de sugerir melhorias?

Se você encontrar algum erro neste material ou tiver alguma sugestão, crítica ou elogio, por favor, envie um e-mail para treinamentos@algaworks.com.

Sobre o autor

Thiago Faria de Andrade (thiago.faria@algaworks.com) é fundador e diretor da AlgaWorks. Iniciou seu interesse por programação aos 14 anos, quando desenvolveu um software para entretenimento e se tornou um dos mais populares no Brasil e outros países de língua portuguesa.

Graduado em Sistemas de Informação e certificado como programador Java pela Sun/Oracle (SCJP), Thiago trabalhou em consultorias para grandes empresas de Uberlândia/MG, co-fundou a OpenK Tecnologia e trabalhou como diretor de tecnologia e treinamentos nesta empresa por quase 6 anos.

Fundou a AlgaWorks após adquirir 100% das cotas da OpenK Treinamentos, e no ano de 2010, possuía experiência profissional de mais de 10 anos, sendo 7 dedicados à tecnologia Java.

Além de programar, Thiago ministra cursos, palestras e presta consultorias sobre Java para órgãos do governo, universidades, pequenas e grandes empresas e pessoas físicas, tendo acumulado aproximadamente 2.500 horas em salas de aulas e auditórios, até o ano de 2010.

Índice

1. Objetivo do curso	9
2. Como aprender Java?	10
2.1. Faça cursos, leia muito e pratique	10
2.2. Assine revistas e acesse fóruns de discussão	10
2.3. Não menospreze os exercícios	10
2.4. Não tenha vergonha em perguntar	10
2.5. Desconecte-se dos aplicativos de mensagens instantâneas	11
2.6. Concentre-se na aula	11
2.7. Evite faltar ou chegar atrasado	11
2.8. Não se assuste com a sopa de letrinhas	11
3. A história do Java	12
4. As plataformas Java	13
4.1. A plataforma Java SE	13
4.2. A plataforma Java EE	13
4.3. A plataforma Java ME	14
4.4. Java Card, JavaFX e Java TV	14
4.5. Como Java evolui?	15
5. Máquina virtual Java	16
5.1. O que é JVM?	16
5.2. A JVM faz o Java ficar lento?	18
6. Baixando, instalando e configurando	20
6.1. Preciso do JDK ou JRE?	20
6.2. Baixando o JDK da Oracle	20
6.3. Instalando o JDK no Windows	22
6.4. Instalando o JDK no Linux	27
7. Fundamentos da linguagem	30
7.1. Vamos instalar a IDE agora?	30
7.2. Codificando o programa “oi mundo”	30
7.3. Compilando e executando	31
7.4. Entendendo o que foi codificado	32
7.5. Erros comuns dos marinheiros de primeira viagem	33
7.6. Comentários	34
7.7. Sequências de escape	35
7.8. Palavras reservadas	36
7.9. Convenções de código	37
7.10. Trabalhando com variáveis	37
7.11. Nomeando variáveis	39
7.12. Operadores aritméticos	40
7.13. Tipos primitivos	41
7.14. Outros operadores de atribuição	44
7.15. Conversão de tipos primitivos	45
7.16. Promoção aritmética	48
7.17. Trabalhando com strings	50
7.18. Recebendo entrada de dados	52
7.19. Operadores de comparação e igualdade	54
7.20. Estruturas de controle if, else if e else	55
7.21. Programar ifs sem abrir e fechar blocos é legal?	57
7.22. Escopo de variáveis	59
7.23. Operadores lógicos	60
7.24. Estrutura de controle switch	63
7.25. Operador ternário	65
7.26. Operadores de incremento e decremento	66
7.27. Estrutura de controle while	68
7.28. Estrutura de controle do/while	68
7.29. Estrutura de controle for	69
7.30. Cláusulas break e continue	70

8. Próximos capítulos	73
9. Sites Interessantes	73
10. Livros Recomendados.....	74

1. Objetivo do curso

As constantes evoluções da tecnologia exigem das pessoas um freqüente aperfeiçoamento de seus conhecimentos profissionais. No dia-a-dia, muitas pessoas não conseguem acompanhar as mudanças e novidades nas tecnologias.

Para estes profissionais, a **AlgaWorks** oferece treinamentos personalizados, ministrados por pessoas com larga experiência no mercado de software. O conteúdo programático de nossos cursos se adequam à realidade, abordando os temas com aulas presenciais realizadas com muita prática.

Tudo isso garante um investimento em TI que se traduz num diferencial competitivo para a empresa, pois sua equipe estará totalmente capacitada a extrair todo seu potencial.

Este curso fornece aos profissionais conhecimentos básicos da linguagem Java, que é utilizada para sedimentar conceitos de orientação a objetos, com exemplos práticos desenvolvidos com a linguagem Java.

Após a conclusão deste curso, você deverá estar apto a:

- Fazer o download das ferramentas necessárias para o desenvolvimento com a linguagem de programação Java;
- Entender e instalar o ambiente de desenvolvimento Java (Java SE);
- Entender e implementar classes e métodos utilizando a linguagem Java;
- Editar, compilar e executar aplicações Java;
- Entender os conceitos de desenvolvimento de aplicações orientado a objetos;
- Entender conceitos avançados de orientação a objetos, como o polimorfismo;
- Escrever um aplicativo Java simples, usando interface gráfica de usuário;
- Escrever programas que implementem os bons conceitos da orientação a objetos, como o encapsulamento.

2. Como aprender Java?

2.1. Faça cursos, leia muito e pratique

Você está no caminho certo para aprender Java e quem sabe se tornar um especialista nesta tecnologia. Fazer cursos, ler bons livros e apostilas são excelentes fontes para aprender qualquer tecnologia, mas só isso provavelmente não será suficiente.

Aprender qualquer nova tecnologia exige dedicação, concentração e muita prática. Por isso, recomendamos fortemente que, ao estudar cada assunto deste curso, você faça todos os exercícios conforme solicitados durante as aulas. Mesmo que você consiga solucionar os exercícios, procure fazer novamente os mesmos exercícios ou outros semelhantes em casa, pois a repetição lhe ajudará bastante a fixar a sintaxe da linguagem e os conceitos realmente importantes.

2.2. Assine revistas e acesse fóruns de discussão

Existem revistas especializadas que falam sobre desenvolvimento em Java, como a Java Magazine (www.javamagazine.com) e MundoJ (www.mundoj.com.br). Nós somos parceiros da MundoJ e indicamos que você a assine, mas se puder, ter as duas revistas chegando em sua casa não é nada mal.

Sempre que tiver uma dúvida, além de poder contar com a ajuda de nossos instrutores, recomendamos que você acesse fóruns de discussão como o www.javafree.org e www.guj.com.br. Provavelmente sua dúvida já foi respondida nestes fóruns, por isso, antes de perguntar, faça uma pesquisa rápida. Se você tiver alguma pergunta que não conseguiu encontrar a resposta, não hesite em perguntar.

2.3. Não menospreze os exercícios

Se você já tiver algum conhecimento em Java ou outras linguagens que possuam a sintaxe parecida, talvez você ache os primeiros exercícios desse curso simples e queira deixar de fazê-los. Nossa dica é: não menospreze os exercícios! Por mais simples que possam parecer para você, é importante que você os faça para descobrir as diferenças, encontrar problemas (e resolvê-los), exercitar sua criatividade e compilar e executar os códigos.

2.4. Não tenha vergonha em perguntar

Tire suas dúvidas quando elas surgirem, não espere outra oportunidade para fazer isso! É comum alguns alunos ficarem com vergonha de fazer alguma pergunta. Nós recomendamos que você faça um esforço para superar esse “medo” e pergunte sempre que tiver alguma dúvida. Nós reconhecemos que ninguém nasce sabendo, e as perguntas dos alunos enriquecem muito mais o curso. Não desperdice a chance de estar com o instrutor disponível para lhe ajudar no que for preciso.

2.5. Desconecte-se dos aplicativos de mensagens instantâneas

Evite perder o foco durante as aulas. O uso de programas de mensagens instantâneas, como MSN, Skype e Google Talk são os grandes vilões para você perder todo o investimento que está fazendo. Se possível, permaneça desconectado de qualquer aplicativo que possa lhe tirar a atenção.

2.6. Concentre-se na aula

Concentre-se no que o instrutor está explicando. Apesar de nos esforçarmos para preparar uma apostila completa para você, o instrutor sempre explicará os assuntos baseados em suas experiências profissionais. Se você desviar a atenção, pode deixar de aprender coisas muito valiosas para o seu futuro. Pode parecer estranho, mas muitos alunos desviam a atenção estudando. Isso mesmo! Enquanto o instrutor explica sobre algum assunto, alguns alunos empolgados podem querer fazer exercícios para se adiantarem ou pesquisar sobre algum recurso da tecnologia na internet. Apesar de ser um bom sinal (pois indica que o aluno aprendeu com facilidade), recomendamos que tome muito cuidado com essa atitude, pois como mencionado anteriormente, você pode estar perdendo informação importante para sua formação.

2.7. Evite faltar ou chegar atrasado

Tente não faltar nas aulas e chegue sempre no horário. Os conteúdos de nossos cursos são bastante intensos, o que quer dizer que, em uma aula de 4 horas, ensinamos muitos conceitos e recursos da tecnologia. Se você perder uma aula ou uma parte dela, terá que se esforçar para recuperar o conteúdo perdido. Apesar de isso ser possível e sempre estarmos disponíveis através de e-mail ou na sala de aula para ajudar você, nada substitui as aulas que ministramos.

No caso de cursos à distância, faça as aulas com regularidade para não ficar muito tempo sem contato com as tecnologias.

2.8. Não se assuste com a sopa de letrinhas

É muito comum, principalmente em Java, as tecnologias serem identificadas por siglas de poucas letras, e talvez essa seja uma grande barreira que as pessoas que querem aprender Java têm que enfrentar.

Quem quer começar a aprender Java é sobrecarregado com siglas como: JDK, JVM, JEE, JSE, JME, J2SE, J2EE, J2ME, J2SDK, JSP, JSF, EL, JSTL, JPA, EJB, JTA, XML, DI, IoC, CoC e outras milhares de letrinhas. Mantenha a calma e não entre em pânico! Concentre-se em aprender cada coisa de uma vez. Você não é obrigado a conhecer todas as siglas para aprender Java, embora no futuro, se você continuar estudando sempre, vai pelo menos saber para que serve cada uma delas.

3. A história do Java



Em 1991, a Sun Microsystems financiou uma pesquisa corporativa interna com o codinome *Green*, acreditando que a próxima área importante seriam os dispositivos eletrônicos inteligentes destinados ao consumidor final. O projeto resultou no desenvolvimento de uma linguagem baseada em C e C++ que seu criador, James Gosling, chamou de Oak (carvalho) em homenagem a uma árvore que dava para a janela do seu escritório na Sun.

Descobriu-se mais tarde que já havia uma linguagem de computador chamada Oak, e quando uma equipe da Sun visitou uma cafeteria local, o nome Java (cidade de origem de um tipo de café importado) foi sugerido e pegou.

O projeto *Green* demonstrou a linguagem através de um controle interativo voltado para TV a cabo. Era muita inovação para a época, e o mercado para dispositivos eletrônicos inteligentes destinados ao consumidor final não estava se desenvolvendo tão rapidamente como a Sun tinha previsto. Pior ainda, um contrato importante pelo qual a Sun competia fora concedido a outra empresa. Então, o projeto estava em risco de cancelamento.

Por pura sorte, a World Wide Web explodiu em popularidade em 1993 e as pessoas da Sun viram o imediato potencial de utilizar Java para criar páginas da Web com o chamado *conteúdo dinâmico (applets)*. Isso deu nova vida ao projeto.

Em maio de 1995, a Sun anunciou Java formalmente em uma conferência importante. Normalmente, um evento como esse não teria gerado muita atenção. Entretanto, Java gerou interesse imediato na comunidade comercial por causa do fenomenal interesse pela World Wide Web.

No ano de 2009, durante uma forte crise financeira, a Oracle comprou a Sun, com promessas de continuar inovando e investindo no Java e demais produtos da Sun.

Atualmente Java está presente em mais de 4,5 bilhões de dispositivos. A plataforma é utilizada para criar páginas da web com conteúdo interativo e dinâmico, para desenvolver aplicativos corporativos de grande porte, aprimorar a funcionalidade de servidores da World Wide Web, fornecer aplicativos para dispositivos destinados ao consumidor final e para muitas outras finalidades.

4. As plataformas Java

O universo Java é composto por uma gama de plataformas, baseadas na idéia de que um software deveria ser capaz de rodar em diferentes máquinas, sistemas e dispositivos. Por diferentes dispositivos entendemos: computadores, servidores, notebooks, handhelds, PDAs (Palm), celulares, cartões inteligentes (*smart cards*), TVs, geladeiras e tudo mais o que for possível.

As principais plataformas do mundo Java são: Java SE, Java EE, Java ME, Java Card, JavaFX e Java TV. Vamos conhecer um pouco de cada uma nos próximos tópicos deste capítulo.

4.1. A plataforma Java SE

A Java SE (Java Platform, Standard Edition) é a versão básica, destinada ao desenvolvimento da maior parte das aplicações desktop que rodam nas estações de trabalho.

Se você está perdido e não sabe o que baixar no site da Oracle para começar a aprender Java, esta é a plataforma que você precisa.

A Java SE é uma rica plataforma que oferece um completo ambiente para o desenvolvimento de aplicações para clientes e servidores. Ela é também a base de outras plataformas.

A Oracle distribui a Java SE na forma de um SDK (Software Development Kit). O pacote do SDK, também conhecido como JDK (Java Development Kit), vem com ferramentas para compilação, execução, *debugging*, geração de documentação (javadoc), empacotador de componentes, bibliotecas comuns e etc.

Neste curso nós usaremos esta plataforma, pois é a base para qualquer desenvolvedor Java.

4.2. A plataforma Java EE

A Java EE (Java Platform, Enterprise Edition) é uma plataforma padrão para desenvolver aplicações Java de grande porte e/ou para a internet, que inclui bibliotecas e funcionalidades para implementar software Java distribuído, baseado em componentes modulares que executam em servidores de aplicações e que suportam escalabilidade, segurança, integridade e outros requisitos de aplicações corporativas ou de grande porte.

A plataforma Java EE possui uma série de especificações (tecnologias) com objetivos distintos, por isso é considerada uma plataforma guarda-chuva. Entre as especificações da Java EE, as mais conhecidas são:

- Servlets: são componentes Java executados no servidor para gerar conteúdo dinâmico para a web, como HTML e XML.
- JSP (JavaServer Pages): uma especialização de Servlets que permite que aplicações web desenvolvidas em Java sejam mais fáceis de manter. É similar às tecnologias como ASP e PHP, porém mais robusta por ter todas as facilidades da plataforma Java.

- JSF (JavaServer Faces): é um framework¹ web baseado em Java que tem como objetivo simplificar o desenvolvimento de interfaces (telas) de sistemas para a web, através de um modelo de componentes reutilizáveis. A proposta é que os sistemas sejam desenvolvidos com a mesma facilidade e produtividade que se desenvolve sistemas desktop (até mesmo com ferramentas que suportam clicar-e-arrastar componentes).
- JPA (Java Persistence API): é uma API padrão do Java para persistência de dados² que usa um conceito de mapeamento objeto-relacional. Essa tecnologia traz alta produtividade para o desenvolvimento de sistemas que necessitam de integração com banco de dados. Só para citar, essa API possibilita que você desenvolva aplicações usando banco de dados sem precisar escrever uma linha sequer de SQL.
- EJB (Enterprise Java Beans): são componentes que executam em servidores de aplicação e possuem como principais objetivos fornecer facilidade e produtividade no desenvolvimento de componentes distribuídos, transacionados, seguros e portáteis.

4.3. A plataforma Java ME

A Java ME (Java Platform, Micro Edition) é uma plataforma que possibilita a criação de sistemas embarcados em dispositivos compactos, como celulares, PDAs, controles remotos, etc.

A plataforma inclui interfaces visuais flexíveis e suporta o desenvolvimento de aplicações seguras e portáteis.

4.4. Java Card, JavaFX e Java TV

A tecnologia Java Card permite que aplicações Java sejam executadas com segurança em cartões inteligentes ou outros dispositivos similares. É muito usada em cartões SIM (chips de telefones celulares) e cartões de bancos. Em cartões SIM, por exemplo, a tecnologia possibilita o armazenamento do identificador do cliente na operadora de telefonia, preferências, agenda de telefones, etc.

A plataforma JavaFX é usada para desenvolver aplicações ricas (no sentido de bonitas e interativas) para a internet, que podem ser executadas em uma ampla variedade de dispositivos conectados, como em ambientes desktop, internet (browser), telefones celulares e TVs.

A Java TV fornece APIs para desenvolvimento de aplicações que são executadas em *set-up boxes* (receptores de TV digital), players de Blu-ray e outros dispositivos de mídia digital.

¹ Framework – um tipo especial de biblioteca de software que fornece APIs para que sejam desenvolvidos sistemas usando uma estrutura padrão para um ambiente de desenvolvimento específico.

² Persistência de dados – códigos desenvolvidos para tratar da persistência de dados são responsáveis pelo armazenamento de informações não-voláteis (recuperáveis), como em bancos de dados ou arquivos.

4.5. Como Java evolui?

A plataforma Java e todas as tecnologias relacionadas evoluem através da força da comunidade mundial.

Para que isso funcione, existe uma entidade chamada JCP (Java Community Process), que organiza e formaliza o processo de definição de uma nova tecnologia ou da atualização de uma tecnologia existente.

Para cada nova idéia de tecnologia ou de atualização, é criada uma JSR (Java Specification Request), que nada mais é que um documento que descreve as propostas de como a tecnologia deve funcionar.

Para iniciar uma JSR, além da idéia inicial do que deve ser feito e qual o problema que se pretende resolver, são nomeados alguns líderes e vários especialistas no assunto, podendo ser pessoas físicas ou empresas. Esses colaboradores conduzem os trabalhos através de reuniões e processos de revisão e votação, até que a versão final seja alcançada.

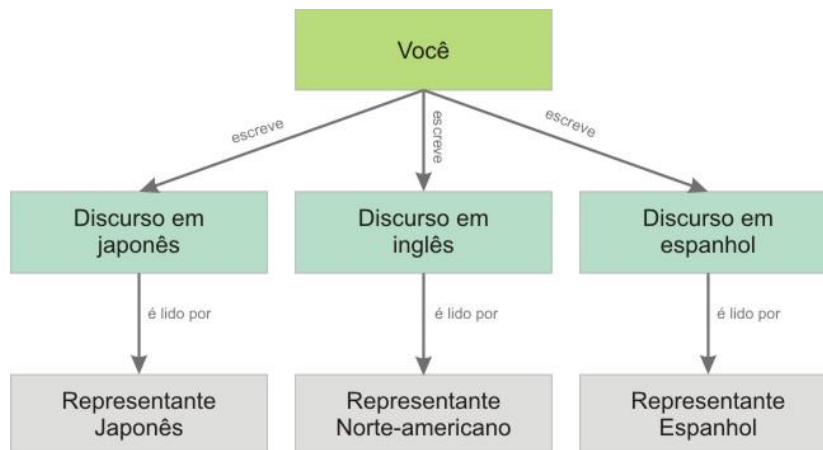
Ao término de uma JSR, tem-se uma documentação completa sobre o funcionamento da tecnologia, uma implementação de referência na forma de código-fonte e um TCK (Technology Compatibility Kit), que é um conjunto de testes que possibilita a validação de um produto desenvolvido para saber se está em conformidade com a especificação (JSR).

Baseadas nas JSRs (especificações), empresas ou comunidades de todo o mundo podem construir produtos (implementações) e oferecer como alternativas aos já existentes, se diferenciando pela qualidade, suporte, documentação e preço.

5. Máquina virtual Java

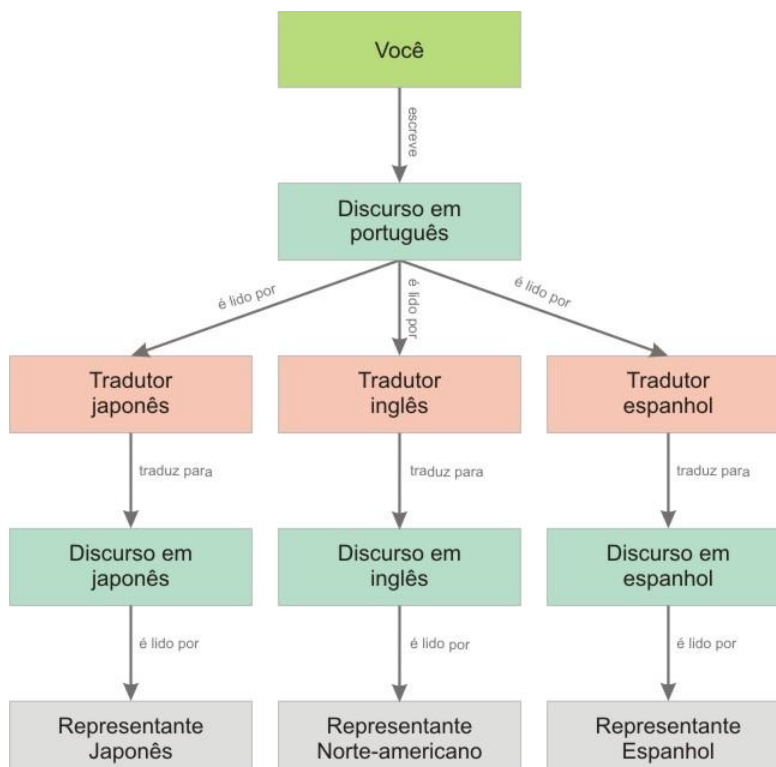
5.1. O que é JVM?

Imagine que você queira escrever um discurso para uma reunião que você vai participar com representantes do Japão, Estados Unidos e Espanha. Esses representantes não sabem falar português, por isso, você terá que aprender a língua deles para se comunicar. É um trabalho possível, porém trabalhoso.



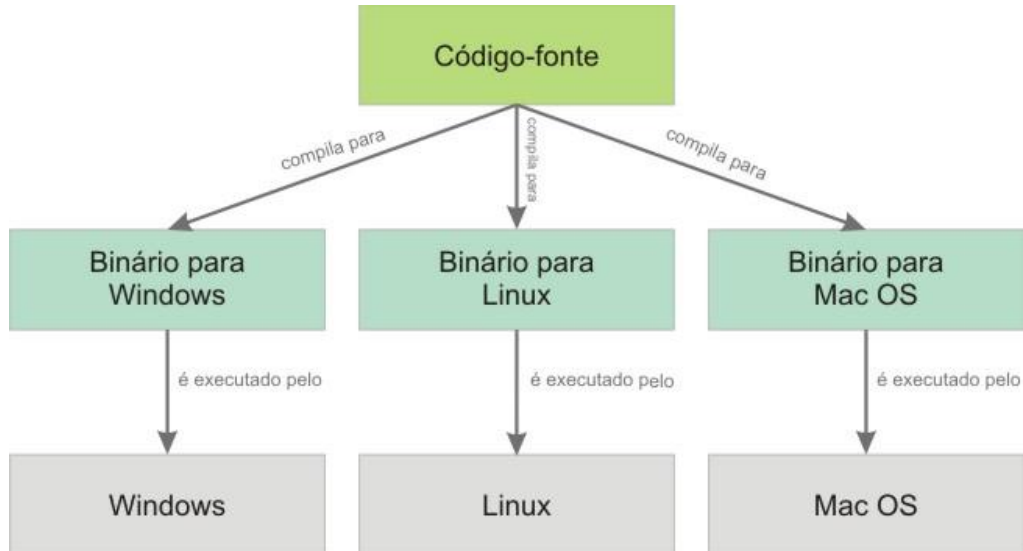
O que você poderia fazer para otimizar seu trabalho? O que acha de contratar tradutores que podem falar o seu idioma mais japonês, inglês ou espanhol?

Se você fizer isso, estará economizando muito do seu tempo, pois precisará focar apenas no seu texto em português, sem se preocupar com traduções e detalhes de cada língua. É como se fosse uma terceirização do trabalho de tradução para idiomas específicos!



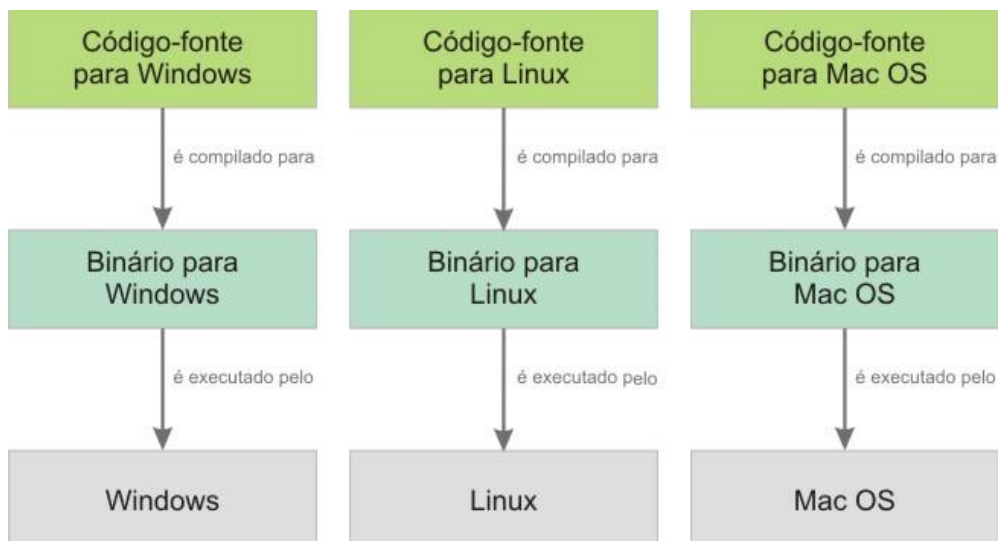
O funcionamento de software funciona quase assim.

Quando você desenvolve e compila um sistema usando a linguagem C, por exemplo, temos o seguinte cenário para um mesmo sistema que deve funcionar em vários sistemas operacionais:



Nessas linguagens de programação, o código-fonte é compilado em código de máquina, que é específico para um SO (sistema operacional) e arquitetura de processador. Isso quer dizer que o binário gerado conhece todos os detalhes do SO para fazer o software funcionar.

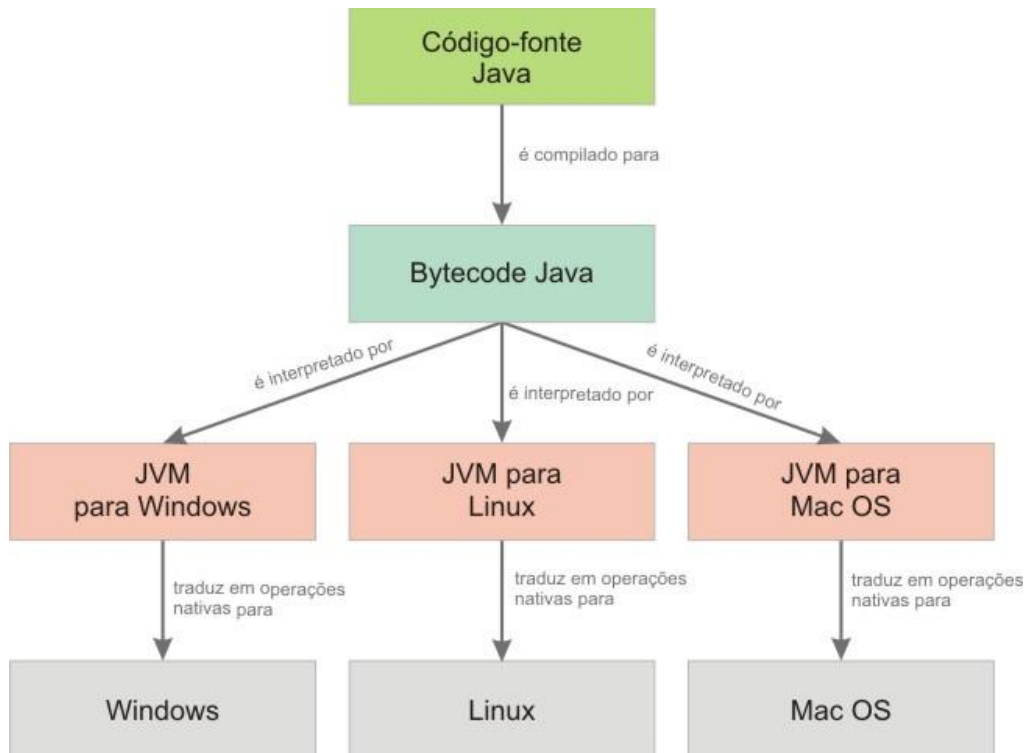
Se quisermos que o software seja executado em múltiplos sistemas operacionais ou com tipos de processadores diferentes, precisamos compilar novamente nosso código-fonte. Parece simples, mas não é só isso! Muitas vezes os programadores usam recursos específicos do SO dentro do código-fonte, como bibliotecas nativas de interfaces gráficas ou de gerenciamento de memória, por exemplo. Se este for o caso, uma simples compilação não resolve o problema, precisando ter múltiplos projetos com códigos-fonte para cada SO.



Com Java, você não precisa se preocupar com nada disso! Graças a JVM (Java Virtual Machine ou Máquina Virtual Java), você pode codificar uma única vez e rodar em diferentes sistemas operacionais.

Quando compilamos um código-fonte Java, um arquivo chamado de *bytecode* é gerado. Ele é um arquivo intermediário que não é legível aos programadores (na verdade alguns nerds conseguem ler) e também não é executável pro si só (o SO não sabe do que se trata esse arquivo).

Apenas a JVM consegue entender o *bytecode* e traduzir as instruções contidas dentro dele em instruções nativas do sistema operacional.



Dessa forma, um único arquivo compilado (*bytecode*) pode ser executado em qualquer sistema operacional, desde que você possua a JVM compatível com ele. Por exemplo, se você quiser executar um sistema desenvolvido no Windows, basta ter uma JVM para Windows instalada em seu computador, mas se tiver um cliente que use Linux, simplesmente instale uma JVM para Linux. Você não precisa fazer mais nada além disso.

Na verdade, a JVM é muito mais que um simples tradutor. Como o próprio nome diz, é uma máquina virtual, ou seja, uma imitação de um computador que consegue gerenciar a pilha de execução, uso de memória, processamento, segurança, etc.

5.2. A JVM faz o Java ficar lento?

Ao contrário do que se parece, a JVM não faz as aplicações desenvolvidas em Java ficarem lentas, mas turbinam elas para que fiquem em alguns casos até mais rápidas que aplicações nativas desenvolvidas em C.

Isso é possível graças ao Hotspot, uma tecnologia de otimização dinâmica que trabalha para aumentar a performance da JVM em tempo de execução.

Quando codificamos um programa na linguagem C, por exemplo, as otimizações são realizadas em tempo de compilação, enquanto em Java isso é feito *Just in Time* (JIT), ou seja, no exato momento que é necessário. Isso é vantajoso, pois a JVM consegue fazer estatísticas de execução do código e otimizar a execução de pontos isolados enquanto a aplicação roda.

6. Baixando, instalando e configurando

6.1. Preciso do JDK ou JRE?

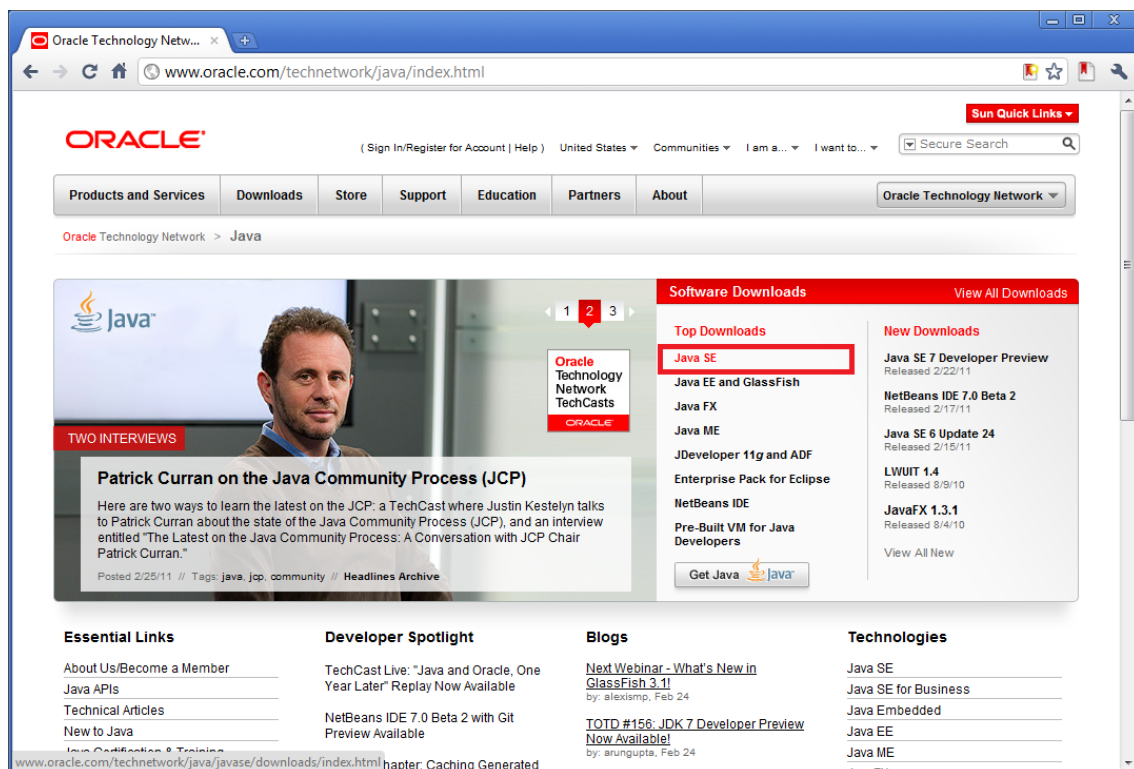
Para desenvolver em Java, precisamos baixar e instalar o JDK (Java Development Kit) da plataforma Java SE. Um kit de desenvolvimento Java possui aplicativos para compilar e debugar seus códigos-fonte, além de diversas outras ferramentas úteis a desenvolvedores de sistemas. O JDK também possui uma JRE (Java Runtime Environment).

JRE é o ambiente de execução da plataforma Java. Usuários de sistemas desenvolvidos em Java precisam instalar apenas a JRE, pois ela possui uma JVM e bibliotecas básicas do Java. Por exemplo, para você usar o software para declaração de imposto de renda desenvolvido pela Receita Federal do Brasil ou para digitar sua senha de acesso em sites de alguns bancos, você precisará do Java instalado. Isso quer dizer que você precisa de pelo menos da JRE, pois o software já foi desenvolvido e ele só precisa ser executado no seu computador.

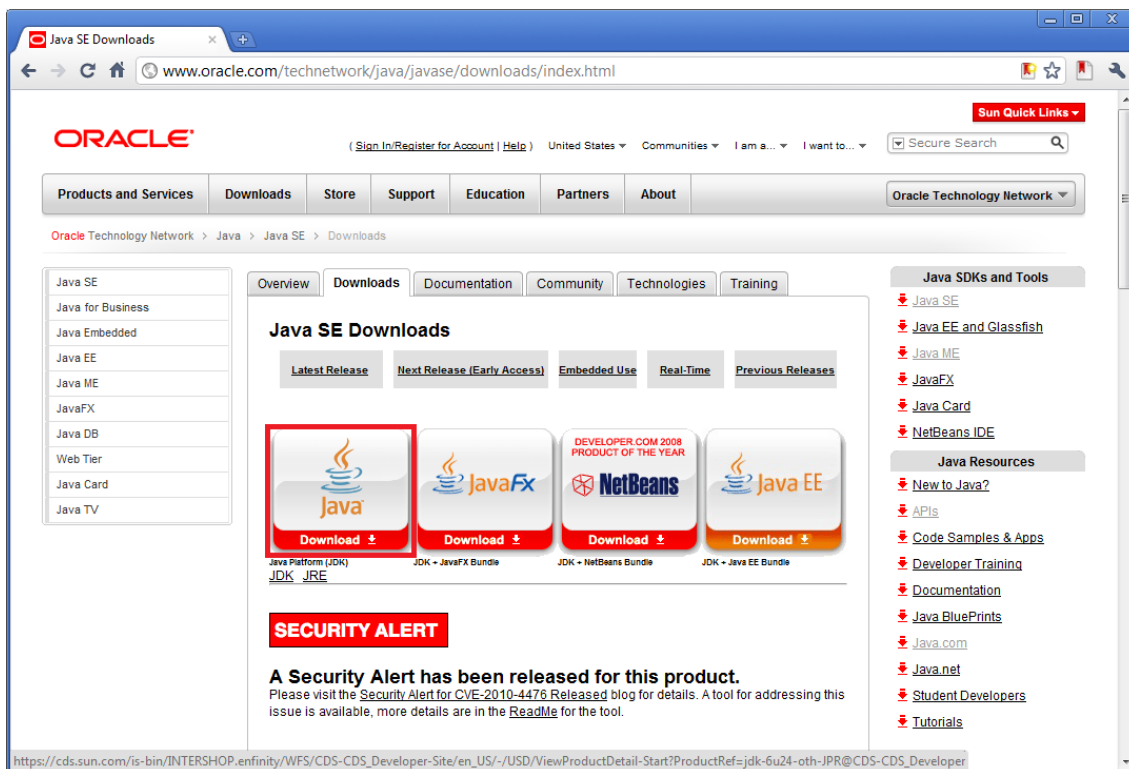
6.2. Baixando o JDK da Oracle

Graças ao modelo que o Java é desenvolvido, através de especificações, o próprio kit de desenvolvimento e a JVM podem ser fornecidos por diferentes empresas.

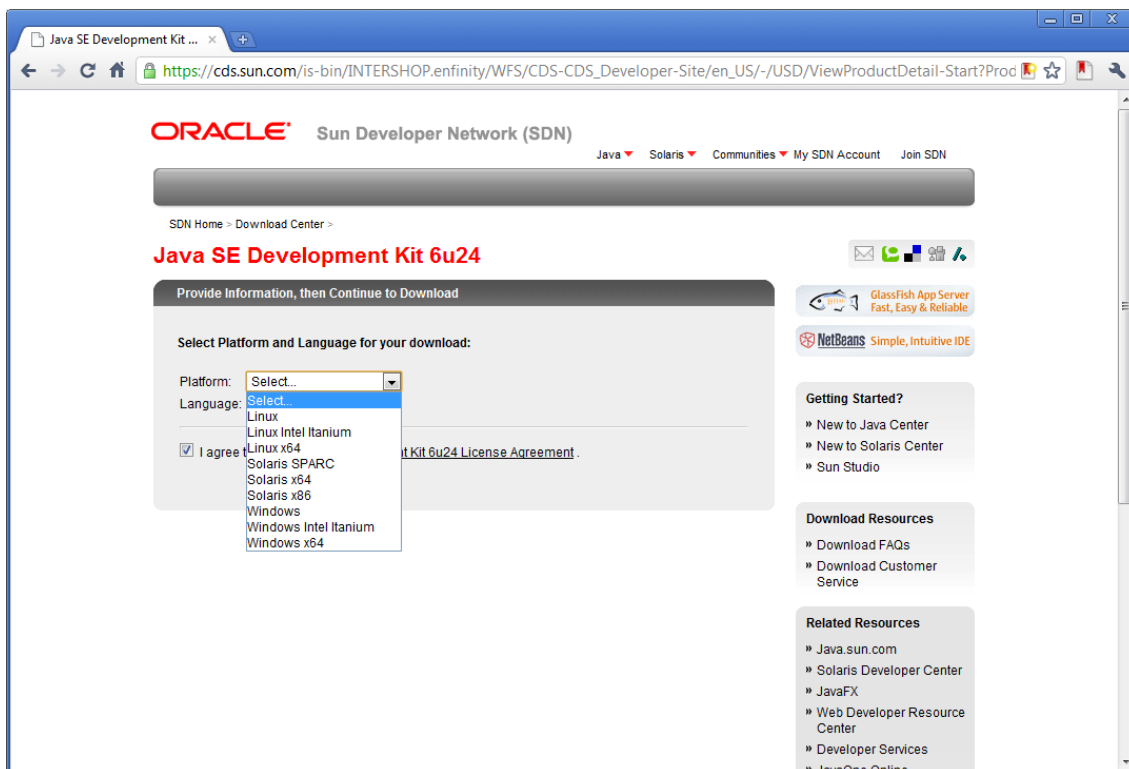
Nós usaremos o JDK da Oracle, que pode ser baixado em <http://java.oracle.com>. Ao acessar este endereço, clique no link “Java SE”, na seção “Top Downloads”.



A Oracle fará sugestões para que você possa baixar também outras plataformas ou a IDE de desenvolvimento NetBeans. Neste momento você não precisa de nada disso, por isso, clique no botão para obter apenas o Java.



Agora você deve selecionar o seu sistema operacional e arquitetura do processador. Se estiver usando um computador com o sistema operacional rodando em 64-bit, você pode selecionar alguma opção que termine com “x64”, mas a opção padrão (32-bit) também deve funcionar. Não esqueça de concordar com a licença da Oracle para fazer o download.



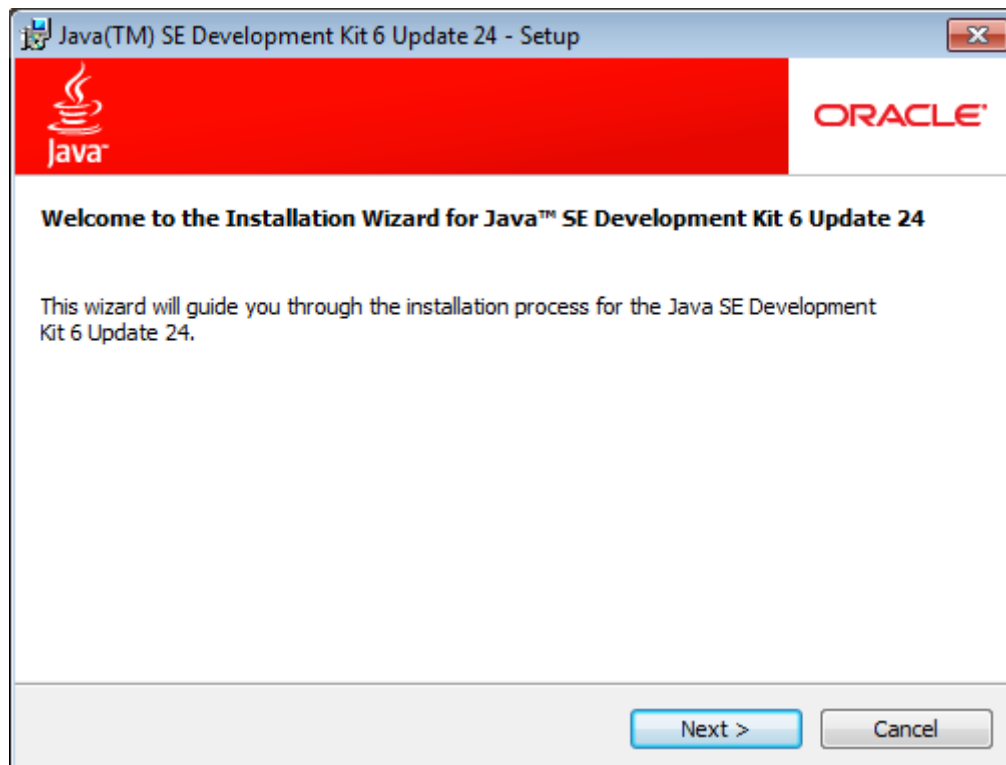
Agora clique no link em destaque para iniciar o download.

File Description and Name	Size
Java SE Development Kit 6u24 jdk-6u24-windows-i586.exe	76.58 MB

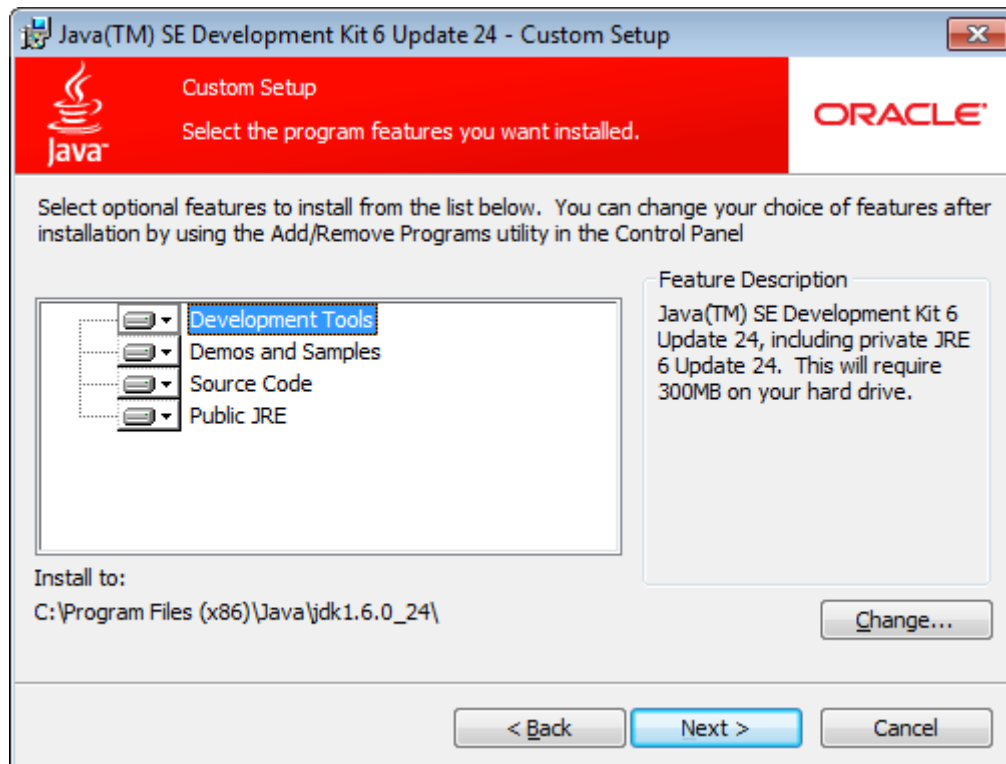
6.3. Instalando o JDK no Windows

Para instalar o JDK no Windows, você já deve ter baixado o arquivo no site da Oracle, conforme explicado na seção anterior. Não vamos cobrir aqui a instalação de kits de desenvolvimento de outros fornecedores.

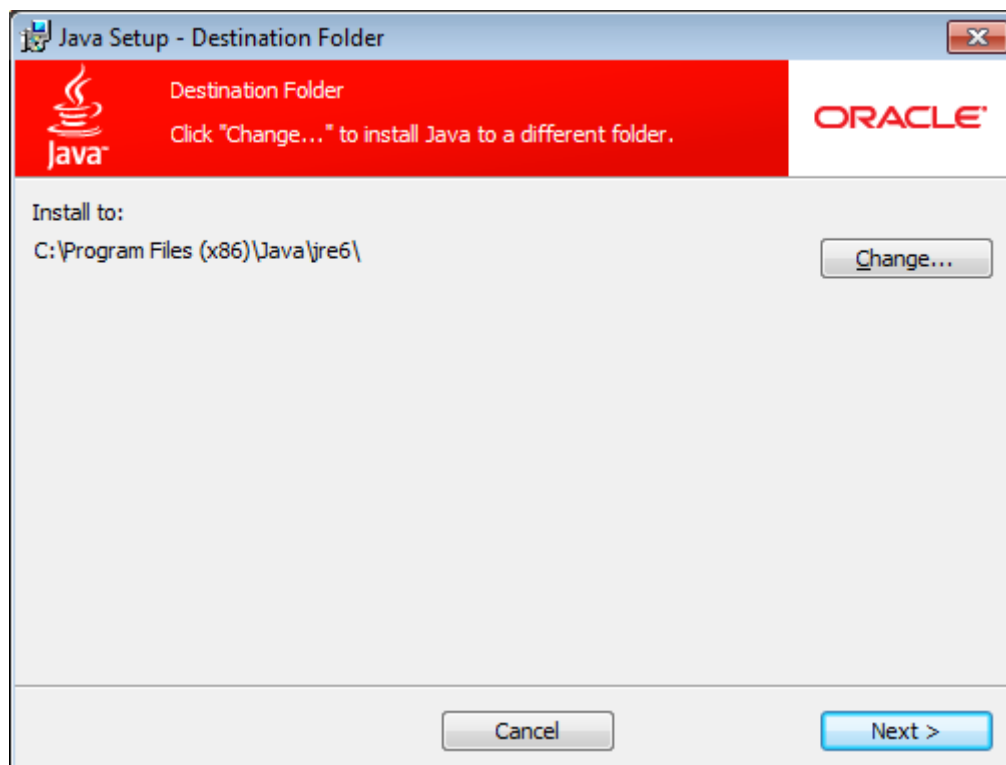
Encontre o arquivo de instalação do JDK e dê um duplo clique nele. O nome desse arquivo poderia ser, por exemplo, “jdk-6uXX-windows-i586.exe”, onde 6 é o número da versão do Java e XX o número do *update* (atualização). Ao abrir a primeira tela, clique em “Next”.



A próxima tela permite selecionar outros recursos que podem ser instalados junto com as ferramentas de desenvolvimento. Clique em “Next” e aguarde o processo de instalação.

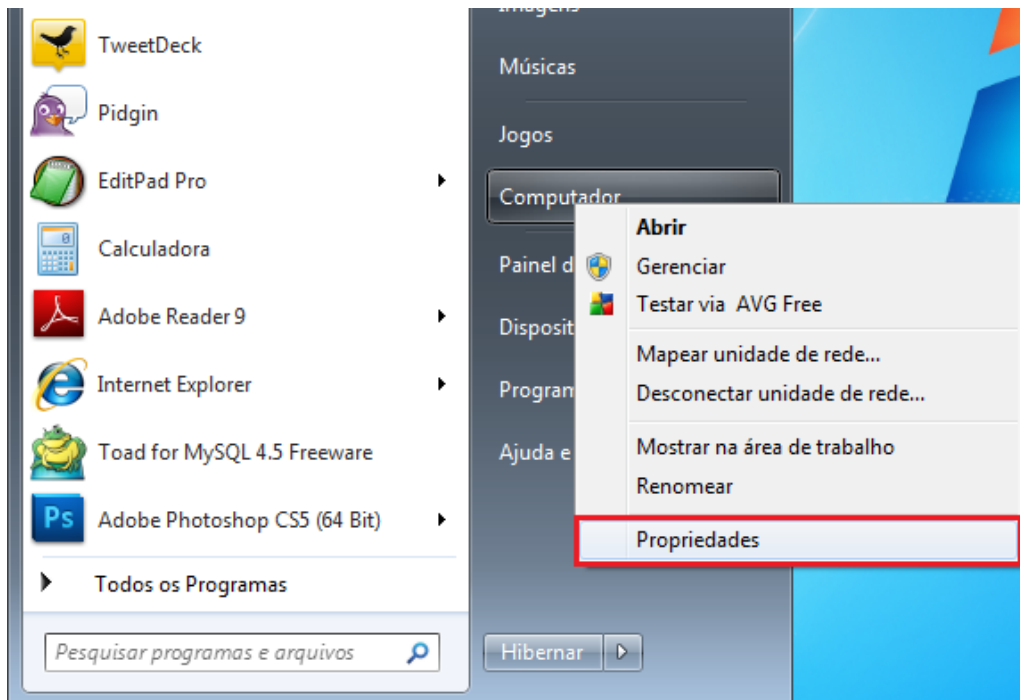


A ferramenta de instalação lhe perguntará se você quer instalar também a JRE. Clique em "Next" e aguarde o término da instalação, depois, clique em "Finish".

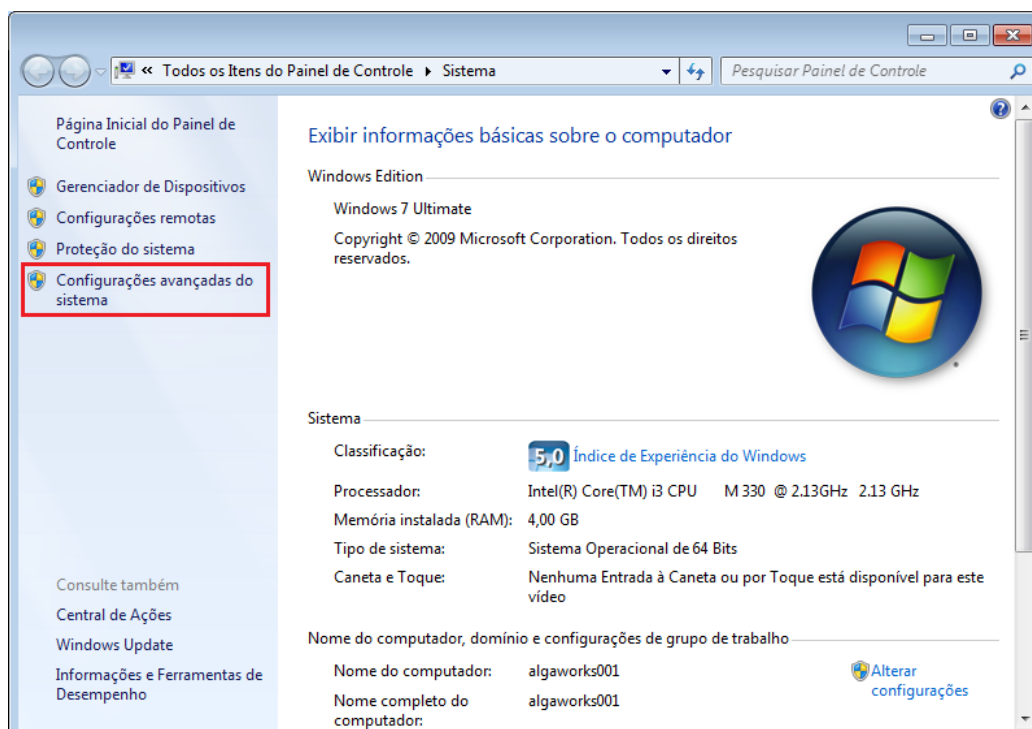


Agora nós precisamos configurar algumas variáveis de ambiente para que as ferramentas de desenvolvimento funcionem adequadamente.

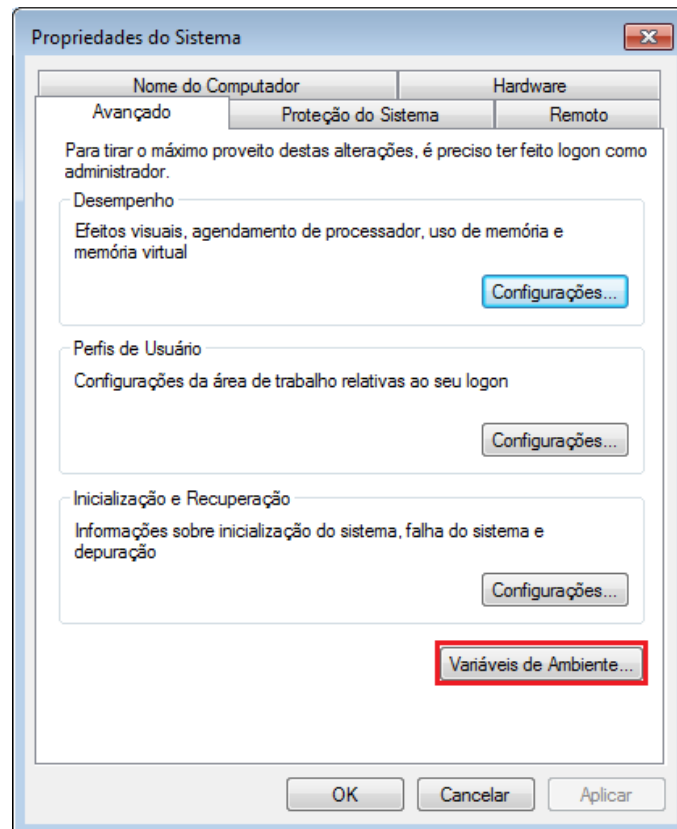
No Windows 7, localize a opção “Computador” no menu “Iniciar” e clique com o botão direito sobre ela. Clique sobre a opção “Propriedades”.



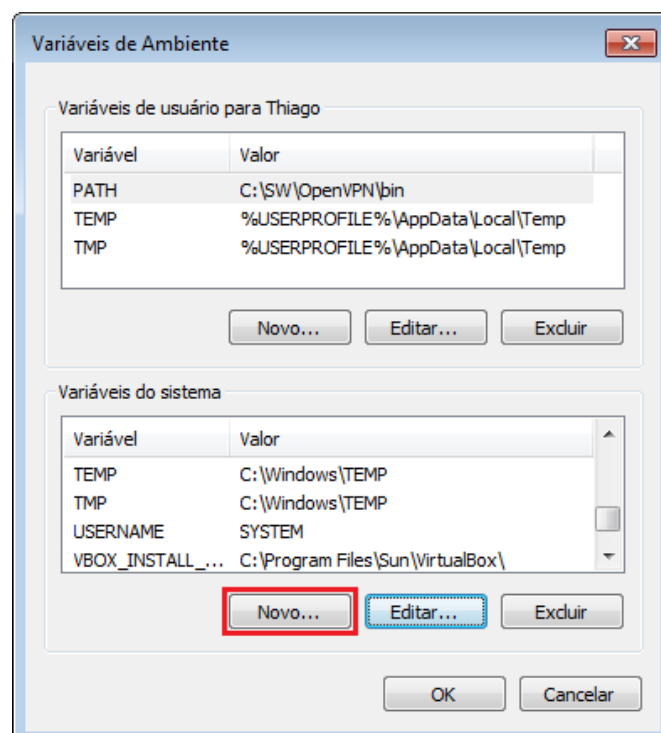
Clique no menu “Configurações avançadas do sistema”, no menu lateral direito.



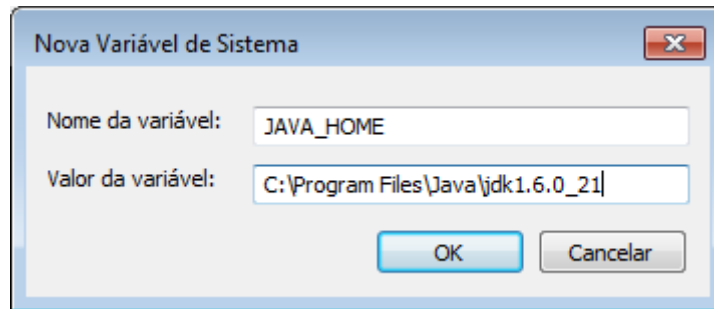
Clique no botão “Variáveis de ambiente”.



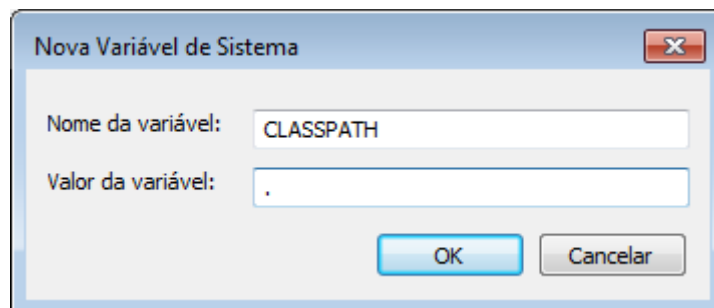
Vamos criar uma nova variável de ambiente do Windows. Para isso, clique no botão “Novo”.



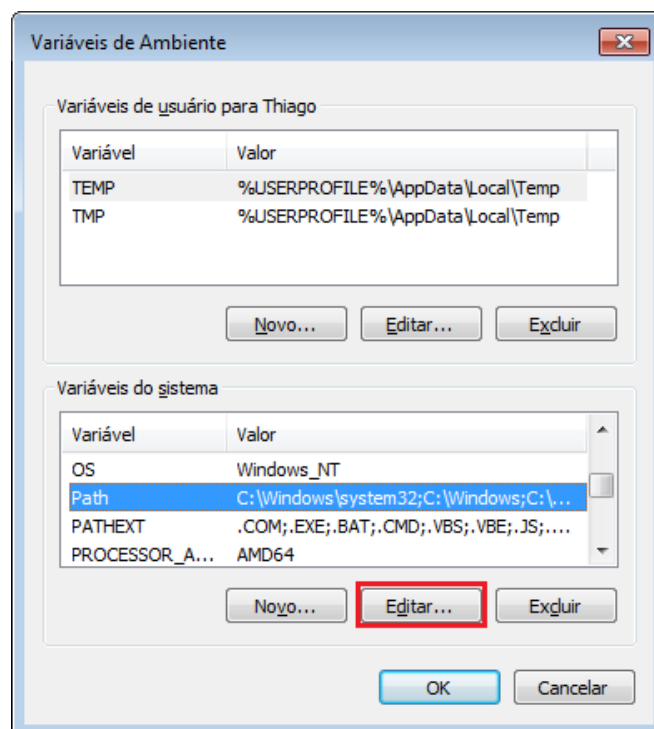
No campo “Nome da variável”, digite `JAVA_HOME`, e no campo “Valor da variável”, digite o caminho onde você instalou o JDK, por exemplo, `C:\Program Files\Java\jdk1.6.0_21`. Clique no botão OK para finalizar a inclusão dessa variável.



Repita esse processo de criação de variável, porém agora com o nome da variável igual a `CLASSPATH` e valor igual a `.` (ponto).

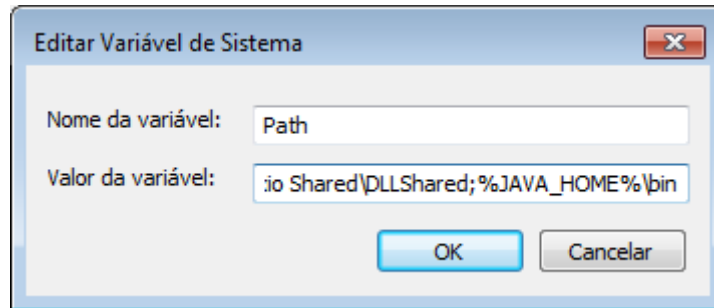


Por último, encontre uma variável já existente chamada `PATH` e clique no botão “Editar”.



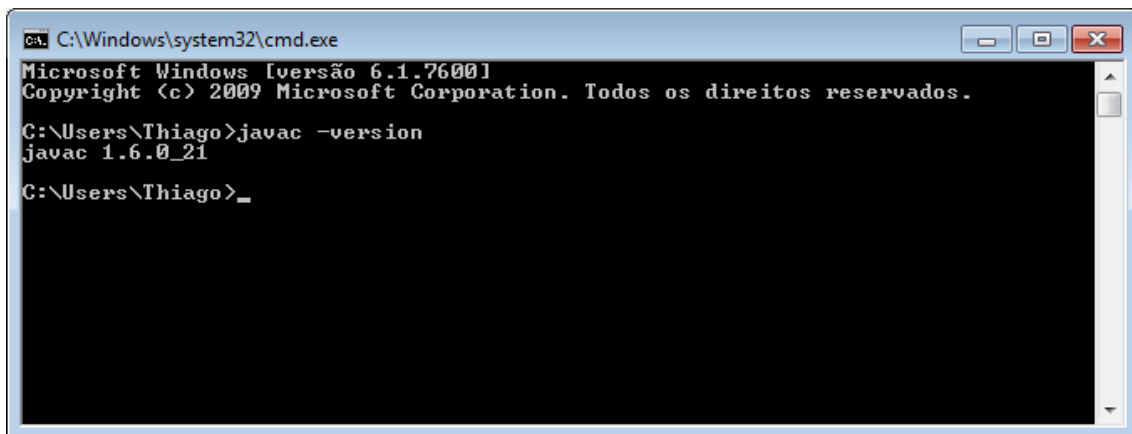
Cuidado para não apagar o conteúdo dessa variável, pois ela é usada pelo sistema operacional.

Inclua `;%JAVA_HOME%\bin` ao final dela. Preste atenção, pois você não pode deixar de incluir o ponto e vírgula antes de `%JAVA_HOME%\bin`.



Se você não esqueceu ou errou nenhum passo, o ambiente básico com o JDK deve estar funcionando. Para conferir, abra o prompt de comando (`cmd.exe`) e digite:

```
javac -version
```



Você deve conseguir visualizar algo parecido com a tela acima.

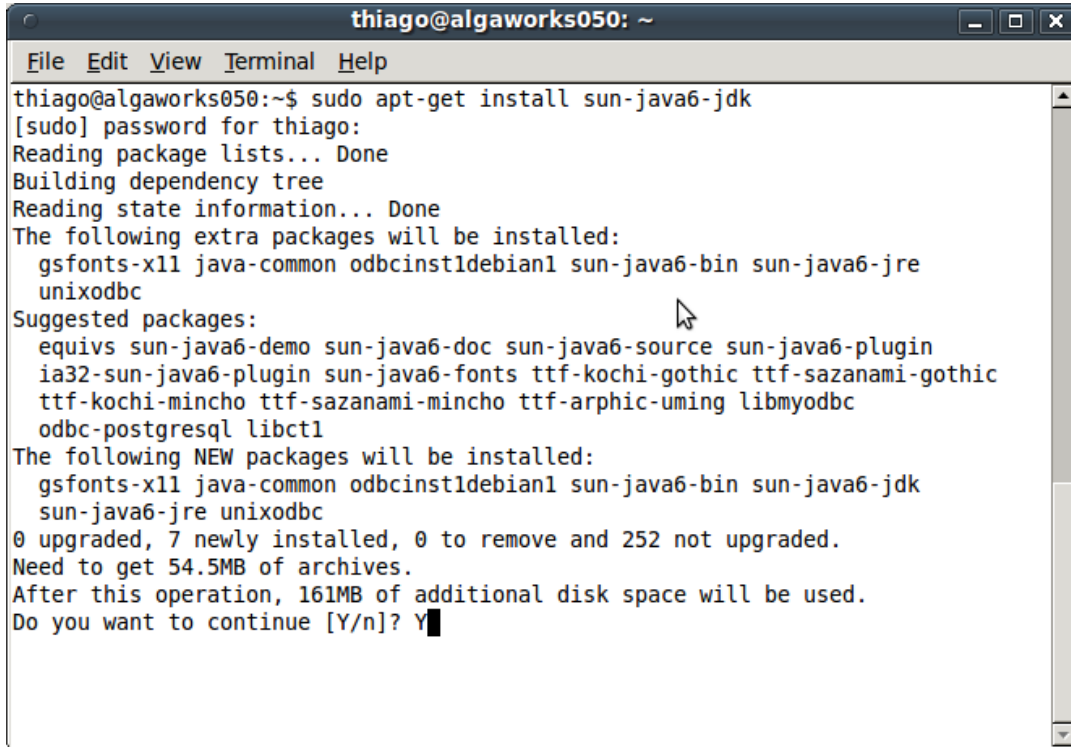
Se receber uma mensagem dizendo que o comando não é reconhecido, refaça todos os passos novamente, pois provavelmente você errou em alguma coisa.

6.4. Instalando o JDK no Linux

O processo de instalação do JDK no Linux depende da distribuição que você usa. No Ubuntu ou outras distribuições similares, a instalação pode ser feita pelo gerenciador de pacotes. Neste caso, você não precisa fazer download do arquivo de instalação do JDK, pois o comando que vamos executar já faz isso através de repositórios que estão na internet.

É muito fácil instalar o JDK da Oracle no Ubuntu. Acesse o terminal e digite:

```
sudo apt-get install sun-java6-jdk
```



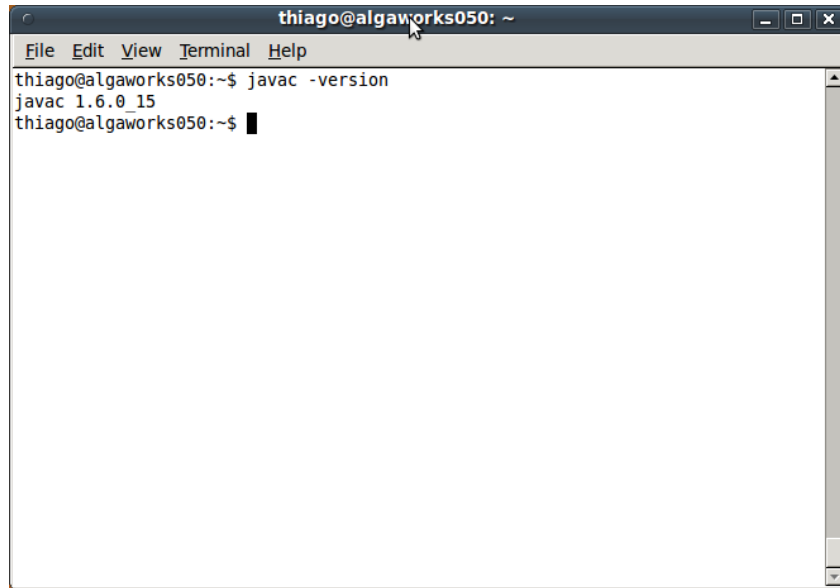
```
thiago@algaworks050: ~  
File Edit View Terminal Help  
thiago@algaworks050:~$ sudo apt-get install sun-java6-jdk  
[sudo] password for thiago:  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following extra packages will be installed:  
  gsfonts-x11 java-common odbcinstldebian1 sun-java6-bin sun-java6-jre  
  unixodbc  
Suggested packages:  
  equivs sun-java6-demo sun-java6-doc sun-java6-source sun-java6-plugin  
  ia32-sun-java6-plugin sun-java6-fonts ttf-kochi-gothic ttf-sazanami-gothic  
  ttf-kochi-mincho ttf-sazanami-mincho ttf-arphic-uming libmyodbc  
  odbc-postgresql libct1  
The following NEW packages will be installed:  
  gsfonts-x11 java-common odbcinstldebian1 sun-java6-bin sun-java6-jdk  
  sun-java6-jre unixodbc  
0 upgraded, 7 newly installed, 0 to remove and 252 not upgraded.  
Need to get 54.5MB of archives.  
After this operation, 161MB of additional disk space will be used.  
Do you want to continue [Y/n]? Y
```

Digite `Y`, tecle *Enter* e aguarde o download. Quando todos os recursos forem baixados, você precisará aceitar os termos da licença para continuar. Selecione a opção “OK” e depois “Yes”. Depois disso, é só aguardar a instalação dos componentes do JDK.

Se o gerenciador de pacotes do Ubuntu dizer que não foi possível encontrar e instalar o JDK, tente digitar os seguintes comandos:

```
sudo add-apt-repository ppa:sun-java-community-team/sun-java6  
sudo apt-get update  
sudo apt-get install sun-java6-jdk
```

Para conferir se está funcionando, digite `javac -version` no terminal. Você deve conseguir visualizar algo parecido com a tela abaixo.



```
thiago@algaworks050: ~  
File Edit View Terminal Help  
thiago@algaworks050:~$ javac -version  
javac 1.6.0 15  
thiago@algaworks050:~$
```

É recomendado que você instale também o pacote com o código-fonte do JDK. Ele será útil para quando estivermos usando uma ferramenta de desenvolvimento (IDE), pois a partir dos códigos-fonte, a IDE conseguirá nos ajudar melhor com os nomes dos parâmetros e as documentações de classes, métodos, etc. Para instalar esse pacote, digite:

```
sudo apt-get install sun-java6-source
```

Se precisar instalar o JDK em outra distribuição do Linux, talvez o gerenciador de pacotes usado pelo sistema operacional possa fazer isso para você. Caso você queira instalar manualmente (em qualquer distribuição, inclusive no Ubuntu), você pode baixar o arquivo de instalação diretamente do site da Oracle e executá-lo. Neste caso, você precisará configurar as variáveis de ambiente, como `JAVA_HOME`, `CLASSPATH` e `PATH`.

7. Fundamentos da linguagem

7.1. Vamos instalar a IDE agora?

Quem está começando a aprender uma nova linguagem, é normal que já queira instalar uma ferramenta para ajudar na edição do código-fonte, mas nós não vamos fazer isso agora.

É necessário que você aprenda a sintaxe da linguagem Java antes de contar com a ajuda de uma IDE. Nós usaremos um editor de texto sem muitos recursos, pois assim você cometerá erros e terá a chance de corrigi-los, o que no processo de aprendizado é fundamental.

Se estiver usando Windows, você poderá usar o famoso bloco de notas, mas caso queira um pouco mais de recursos (como abrir vários arquivos em abas e visualizar o código-fonte com destaques em cores), recomendamos o Notepad++ (<http://notepad-plus-plus.org>).

Fique tranquilo, pois quando precisamos de mais produtividade, iremos apresentar ferramentas para desenvolvimento e adotar uma para continuar os estudos.

7.2. Codificando o programa “oi mundo”

Quando você aprende qualquer linguagem de programação, normalmente o primeiro programa que desenvolve é o mais simples possível. Na maioria das vezes ele é chamado de “Olá mundo” ou “Hello World”. Para fazer um pouco diferente, chamamos nosso primeiro programa de “Oi Mundo”.

O primeiro programa é importante para aprendermos a estrutura básica que usaremos nos próximos exemplos, o processo de compilação e execução e também para resolver erros comuns de principiantes.

Veja abaixo o código-fonte de um programa simples que imprime na saída padrão (console) a mensagem “Oi mundo”.

```
class OiMundo {  
  
    public static void main(String[] args) {  
        System.out.println("Oi mundo");  
    }  
  
}
```

Para começar a programar em Java, crie uma pasta para você colocar todos os exemplos e exercícios do curso. Inicie o seu editor de texto favorito (como o Notepad++) e digite o código do exemplo acima.

Por enquanto, não se preocupe com o significado do que escrevemos, pois você aprenderá cada coisa na hora certa.

Você deve digitar o código exatamente como foi apresentado no exemplo, inclusive letras maiúsculas e minúsculas, pois a linguagem Java é *case-sensitive*.

Quando terminar de digitar tudo, salve o arquivo com o nome `OiMundo.java`. Preste atenção novamente às letras maiúsculas e minúsculas do nome do arquivo e também na extensão, que deve ser `.java`.

7.3. Compilando e executando

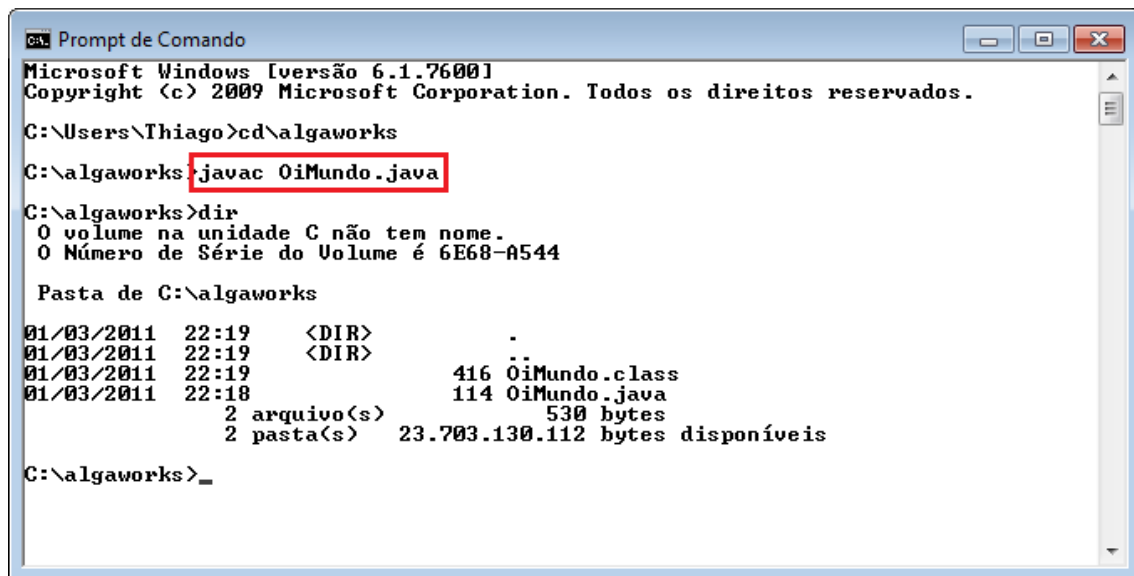
Agora vamos compilar nosso primeiro programa. O processo de compilação é o que transforma o código-fonte em *bytecode* (aquele que só a JVM consegue interpretar).

Entre no prompt de comando do Windows (ou terminal no Linux), acesse a pasta onde você salvou o arquivo `OiMundo.java` e digite:

```
javac OiMundo.java
```

O `javac` é o programa do JDK responsável por compilar um arquivo com código-fonte Java. Se funcionar, o programa ficará silencioso (não aparecerá nenhuma mensagem de sucesso). Se der alguma coisa de errado, você ficará sabendo, pois podem surgir várias mensagens estranhas no terminal (quando você aprender melhor, não será mais tão estranha assim).

Para confirmar se o arquivo foi compilado, você pode listar todos os arquivos da pasta usando o comando `dir` se for Windows ou `ls` se for Linux. Se um novo arquivo chamado `OiMundo.class` aparecer, é porque você teve sucesso ao compilar seu primeiro programa.



```

C:\Users\Thiago>cd\algaworks
C:\algaworks>javac OiMundo.java
C:\algaworks>dir
O volume na unidade C não tem nome.
O Número de Série do Volume é 6E68-A544

Pasta de C:\algaworks
01/03/2011  22:19    <DIR>          .
01/03/2011  22:19    <DIR>          ..
01/03/2011  22:19                416 OiMundo.class
01/03/2011  22:18                114 OiMundo.java
                2 arquivo(s)          530 bytes
                2 pasta(s) 23.703.130.112 bytes disponíveis

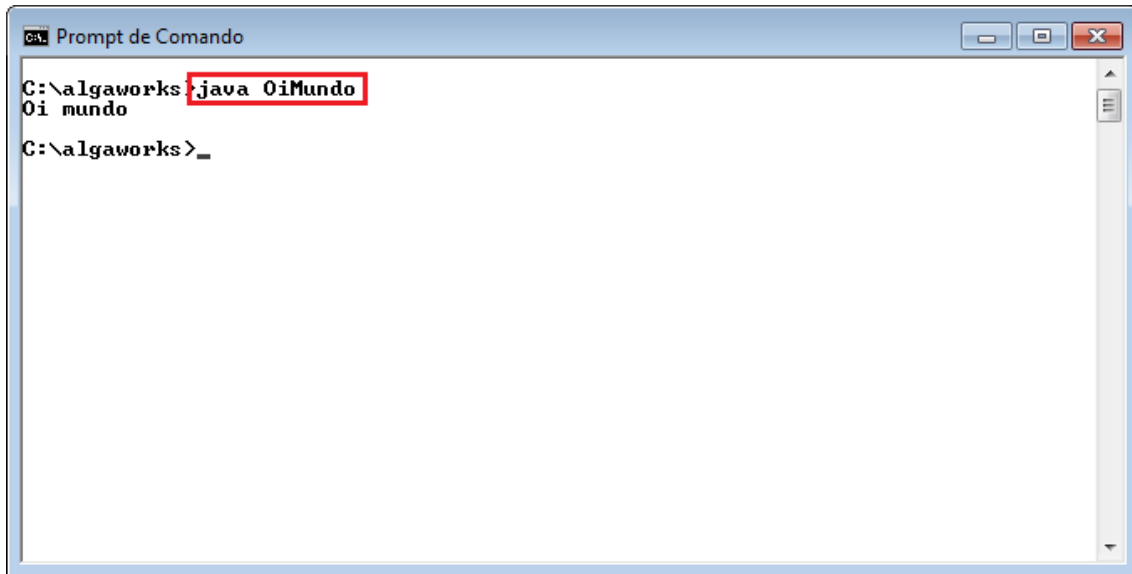
C:\algaworks>_
```

O arquivo com extensão `.class` é o *bytecode* gerado (executável). Se você for curioso (a), poderá tentar abri-lo usando um editor de texto. Como disse, só a JVM consegue interpretá-lo (além de, claro, alguns nerds de plantão).

Agora que você já tem o programa compilado, para executá-lo, digite o comando:

```
java OiMundo
```

Preste atenção novamente para as letras maiúsculas e minúsculas.



```
C:\algaworks>java OiMundo
Oi mundo
C:\algaworks>
```

Se funcionar, você deve ver a mensagem “Oi mundo” no seu terminal, como a imagem acima.

Apenas para ter certeza que você entendeu, o que acabamos de executar no último passo foi o arquivo `OiMundo.class` (mas veja que não podemos colocar a extensão dele quando vamos executá-lo). Você poderia ter apagado ou movido o arquivo `OiMundo.java` que mesmo assim a execução teria sucesso, pois nesse momento apenas o *bytecode* é lido e executado.

7.4. Entendendo o que foi codificado

Vamos estudar um pouco sobre o que codificamos no primeiro programa. Não entraremos em detalhe em tudo para não confundir, mas fique tranquilo que você aprenderá tudo ainda neste curso.

A primeira linha do arquivo declarou um nome de classe. Como ainda não estudamos o que é uma classe (veremos isso em Orientação a Objetos), chamaremos de “programa”, portanto, a primeira linha declarou o programa com o nome “OiMundo”.

A abertura e fechamento das chaves indicam um bloco de código. Tudo que estiver lá dentro pertence ao programa `OiMundo`.

```
class OiMundo {  
  
}
```

Dentro do bloco de código do programa, criamos um método chamado `main`. Como ainda não estudamos o que é um método em orientação a objetos, podemos chamá-lo de procedimento ou função (como preferir).

O método `main` é necessário para que nosso programa seja executado quando digitamos o comando `java OiMundo`. Este método é o ponto de entrada do programa, por isso, ele será executado automaticamente quando solicitarmos a execução do programa.

O nome do método não pode ser alterado, pois apesar de não pertencer à sintaxe da linguagem, é um padrão do Java que significa um ponto inicial de um programa desenvolvido.

O bloco de código delimitado pelas chaves deve possuir uma ou mais linhas com a programação do que o sistema deve fazer.

```
public static void main(String[] args) {  
}
```

Em nosso exemplo, nosso programa apenas imprime “Oi mundo” na tela. Para fazer isso, usamos o método `System.out.println`.

Todo texto (string) em Java é delimitado por aspas duplas, e toda instrução (comando) deve terminar com um ponto e vírgula. Note também que o texto “Oi mundo” está entre parênteses, que indica o início e término de um parâmetro do método.

```
System.out.println("Oi mundo");
```

O nome do arquivo precisou ser exatamente `OiMundo.java`. Em Java, o nome do programa (classe) deve coincidir com o nome do arquivo, portanto, se seu programa chamasse `QueroAprenderJava`, o seu arquivo com o código-fonte deveria ter o nome `QueroAprenderJava.java`.

7.5. Erros comuns dos marinheiros de primeira viagem

Marinheiros de primeira viagem costumam cometer erros comuns, pois a linguagem Java é bastante restritiva. Vejamos alguns erros que você pode cometer e como corrigi-los:

1. Ao compilar, aparece o erro:

```
OiMundo.java:3: cannot find symbol  
symbol   : class string
```

Você digitou `string` com a letra “s” em minúsculo. O correto é:

```
public static void main(String[] args)
```

2. Ao compilar, aparece o erro:

```
package system does not exist
```

Você digitou `system` com a letra “s” em minúsculo. O correto é:

```
System.out.println("Oi mundo");
```

3. Ao compilar, aparece o erro:

```
';' expected
```

Você esqueceu de finalizar a instrução com um ponto e vírgula. Veja:

```
System.out.println("Oi mundo");
```

4. Ao compilar, aparece o erro (ou algo parecido):

```
class x is public, should be declared in a file named x.java
```

Você mudou o nome do programa, mas não mudou o nome do arquivo adequadamente, ou esqueceu de colocar as iniciais do nome do programa em letras maiúsculas:

```
class OiMundo
```

5. Ao executar, aparece o erro (ou algo parecido):

```
Exception in thread "main" java.lang.NoClassDefFoundError:  
x (wrong name: X)
```

Você digitou o nome do programa sem levar em consideração as letras maiúsculas e minúsculas. Digite o comando `java OiMundo` com as iniciais do nome do programa usando letras maiúsculas maiúsculas.

6. Ao compilar, aparece o erro (ou algo parecido):

```
Exception ... "main" java.lang.NoClassDefFoundError: OiMundo
```

A variável de ambiente `CLASSPATH` de seu computador não está configurada ou está incorreta. Verifique se executou todos os passos de instalação e configuração corretamente.

7.6. Comentários

Comentários são textos que podem ser incluídos no código-fonte normalmente para descrever como determinado programa ou bloco de código funciona. Os comentários são ignorados pelo compilador, por isso, eles não modificam o comportamento do programa.

Em Java, você pode comentar blocos de códigos inteiros ou apenas uma linha. Para comentar uma única linha, faça como no exemplo a seguir:

```
class OiMundo {  
  
    public static void main(String[] args) {  
        // imprime uma mensagem na saída padrão  
        System.out.println("Oi mundo");  
    }  
}
```

Para comentar um bloco de código, use `/* */` para abrir e fechar. Veja um exemplo:

```
class OiMundo {  
  
    public static void main(String[] args) {  
        /*  
        Esta linha será ignorada pelo compilador.  
        System.out.println("Esta instrução será ignorada também");  
        E esta linha também. */  
        System.out.println("Oi mundo");  
    }  
  
}
```

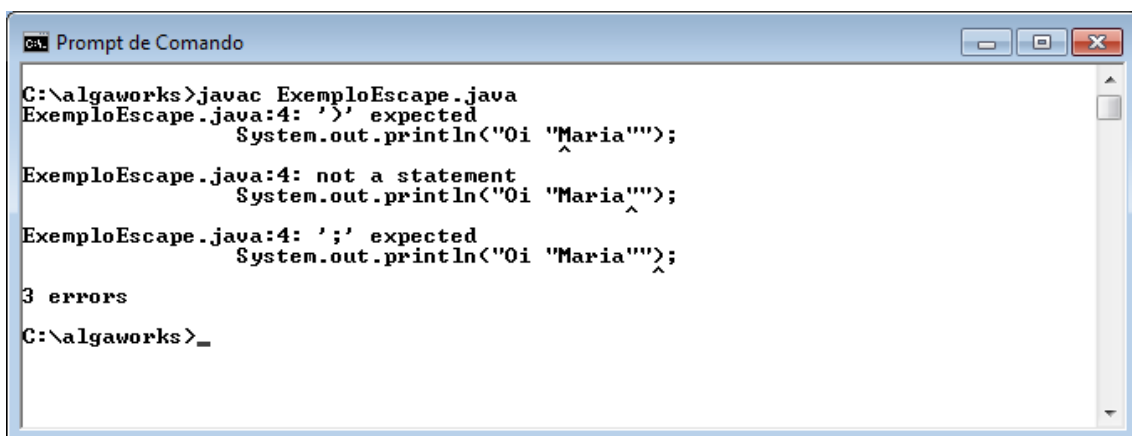
7.7. Seqüências de escape

Seqüências de escape são combinações de caracteres iniciadas por \ (contra barra) e usadas para representar caracteres de controle, aspas, quebras de linha, etc.

Para ficar melhor entendido, vamos fazer um teste. O exemplo abaixo tenta imprimir a mensagem Oi "Maria" (com o nome "Maria" entre aspas duplas).

```
class ExemploEscape {  
  
    public static void main(String[] args) {  
        System.out.println("Oi "Maria");  
    }  
  
}
```

Ao tentarmos compilar, temos um belo erro (não tão belo assim):



```
CA. Prompt de Comando  
  
C:\algaworks>javac ExemploEscape.java  
ExemploEscape.java:4: '}' expected  
        System.out.println("Oi "Maria");  
                                ^  
ExemploEscape.java:4: not a statement  
        System.out.println("Oi "Maria");  
                                ^  
ExemploEscape.java:4: ';' expected  
        System.out.println("Oi "Maria");  
                                ^  
3 errors  
C:\algaworks>_
```

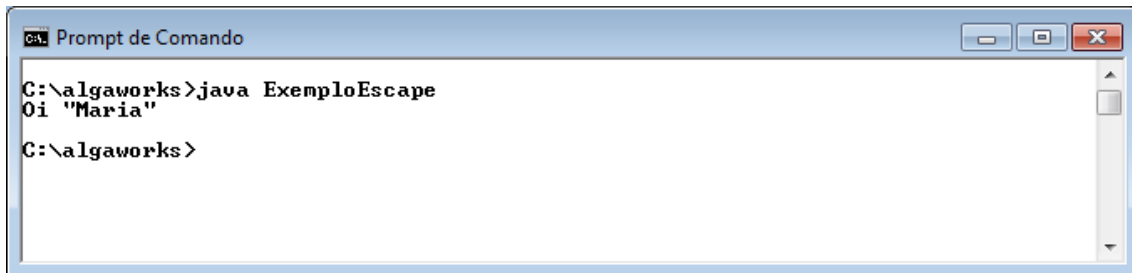
O problema aqui é que o compilador Java não conseguiu entender o que significa a palavra "Maria". Se você prestar atenção, notará que a segunda aspa fechou a primeira, e a quarta fechou a terceira logo após o nome "Maria". Você percebeu que não existem aspas envolvendo o nome "Maria"?

E se quisermos dizer ao compilador que “Maria” é um texto, mas que as aspas que envolvem o nome também são textos? Eis que usamos uma sequência de escape.

```
System.out.println("Oi \"Maria\"");
```

A sequência `\"` diz ao compilador que a aspa deve ser considerada como um texto, e não como um delimitador de String. Agora nosso programa deve compilar e rodar normalmente.

Ao executar nosso programa, as aspas que envolvem o nome aparecem na saída padrão:



Existem várias outras sequências de escape, sendo que as mais conhecidas são:

- `\n` para nova linha
- `\\` para uma barra invertida
- `\"` para aspas duplas

7.8. Palavras reservadas

As palavras-chave são palavras especiais, reservadas em Java, que têm significados para o compilador, que as usa para determinar o que seu código-fonte está tentando fazer. Você não poderá usar as palavras-chave como identificadores (nomes) de classes, métodos ou variáveis. As palavras-chave reservadas estão listadas a seguir:

abstract	boolean	break	byte	case	catch
char	class	const	Continue	default	do
double	else	extends	Final	finally	float
for	goto	if	Implements	import	instanceof
int	interface	long	Native	new	package
private	protected	public	Return	short	static
strictfp	Super	switch	synchronized	this	throw
throws	transient	try	Void	volatile	while
assert	enum				

Você aprenderá a maioria das palavras-chave da listagem acima no momento que for adequado, por isso, não se preocupe em memorizá-las.

Apesar de `goto` e `const` serem palavras-chave reservadas, elas não possuem significado para o compilador. Na verdade, se você tentar usá-las, receberá um erro ao compilar seu código-fonte.

7.9. Convenções de código

Ao desenvolver em Java, encorajamos você a usar as convenções de codificação da linguagem Java, fornecido oficialmente pela Oracle. Esse padrão é usado internamente pela Oracle no desenvolvimento do próprio Java e por praticamente todas as empresas do mundo que desenvolver algum software sério em Java.

O documento completo, que aborda apenas forma de escrita, nomes de arquivos, classes, variáveis, indentação, comentários, métodos e organização do código-fonte, pode ser lido em:

<http://www.oracle.com/technetwork/java/codeconv-138413.html>

As vantagens em utilizar esse padrão é que, além de você estar programando usando o mesmo estilo de código-fonte de todo o mundo, as convenções também facilita a manutenção no futuro por outros desenvolvedores e até mesmo por você, pois aumenta a legibilidade do código-fonte.

Sempre que acharmos necessário, citaremos algumas convenções de código para você já ir se acostumando.

Para começar, é muito comum, praticamente uma regra, que os nomes de seus programas (classes) usem uma prática chamada *Camel/Case*. Isso significa que as iniciais das palavras em uma frase que nomeia seu programa devem ser iniciadas com letras maiúsculas e unidas sem espaços. Por exemplo, se você tiver um programa responsável por gerar notas fiscais, um nome válido seria `GeradorNotaFiscal`, mas não seria correto, de acordo com esta prática, usar os nomes `Geradornotafiscal`, `geradornotafiscal` ou `Gerador_Nota_Fiscal`. Veja:

```
class GeradorNotaFiscal {  
}
```

7.10. Trabalhando com variáveis

Em linguagens de programação, variáveis são nomes simbólicos dados a informações alocadas na memória do computador. As variáveis são definidas, atribuídas e acessadas ou calculadas através do código-fonte do programa. Durante a execução de um programa, os conteúdos das variáveis podem mudar através de algum processamento.

Em Java, as variáveis devem ser declaradas com um tipo fixo para poder ser usadas. Isso quer dizer que, uma variável do tipo inteiro, por exemplo, não poderá ser alterada para um tipo real (decimal).

Para começar, vamos declarar uma variável chamada `quantidade` do tipo `int`. O tipo `int` é capaz de armazenar apenas valores inteiros negativos ou positivos. Veja:

```
int quantidade; // declarando variável inteiro
```

A variável acima foi apenas declarada, isso quer dizer que não atribuímos nenhum valor para ela. Para fazermos isso, usamos o operador `=` (igual) seguido por um número inteiro.

```
quantidade = 10; // atribuindo o valor 10
```

Agora a variável `quantidade` possui o valor 10. Se precisarmos alterar o valor da variável `quantidade`, podemos atribuir um novo valor a ela. Vamos dizer que a variável deve possuir o valor 15.

```
quantidade = 15; // atribuindo o valor 15
```

É muito comum precisarmos mostrar o valor de uma variável na tela do usuário. Para fazermos isso de uma forma bem simples, podemos usar `System.out.println` passando como parâmetro o nome da variável.

```
// imprimindo o valor da variável  
System.out.println(quantidade);
```

Veja como ficou este exemplo completo. Incluímos mais uma instrução de impressão da variável `quantidade` entre a atribuição do valor 10 e do valor 15 para podermos ver o valor da variável antes de depois de ser modificada.

```
class Variaveis1 {  
  
    public static void main(String[] args) {  
        int quantidade;  
        quantidade = 10;  
        System.out.println(quantidade);  
        quantidade = 15;  
        System.out.println(quantidade);  
    }  
}
```

Se você quiser economizar uma linha de código, pode declarar e atribuir a variável na mesma linha, como no exemplo abaixo:

```
int quantidade = 10; // declarando e atribuindo
```

Veja como ficaria no exemplo completo.

```
class Variaveis1 {  
  
    public static void main(String[] args) {  
        int quantidade = 10;  
        System.out.println(quantidade);  
        quantidade = 15;  
        System.out.println(quantidade);  
    }  
}
```

7.11. Nomeando variáveis

As variáveis em Java podem conter letras, dígitos, `_` (*underscore*) e `$` (dólar), porém elas não podem ser iniciadas por um dígito e não podem ser palavras reservadas.

Veja alguns nomes de variáveis válidos:

```
int quantidade; // pode ser toda em letras minúsculas  
int quantidade_alunos; // pode ter underscore  
int QUANTIDADE; // pode ser toda em letras maiúsculas  
int QuantidadeAlunos; // pode ter letras maiúsculas e minúsculas  
int $quantidade; // pode iniciar com dólar  
int _quantidade; // pode iniciar com underscore  
int quantidade_alunos_nota_10; // pode ter dígitos
```

Agora alguns nomes de variáveis inválidos (que nem compila):

```
int 2alunos; // não pode iniciar com dígitos  
int quantidade alunos; // não pode ter espaços  
int new; // new é uma palavra reservada do Java
```

Apesar de a linguagem suportar letras maiúsculas e minúsculas, *underscore*, dólar e dígitos nos nomes das variáveis, a convenção de código Java diz que elas devem ser nomeadas com a inicial em letra minúscula e as demais iniciais das outras palavras em letras maiúsculas. Veja alguns exemplos:

```
int quantidadeAlunos;  
int quantidadeAlunosNota10;  
int numeroDeAlunosAprovados;  
int totalAlunosReprovados;
```

As declarações de variáveis abaixo estão corretas para o compilador, mas não está de acordo com o padrão de código usado mundialmente, por isso, evite-as a todo custo.

```
int quantidade_alunos;  
int QuantidadeAlunosNota10;  
int NUMERODEALUNOSAPROVADOS;  
int Total_Alunos_Reprovados;
```

É uma boa prática escrever as palavras completas quando vamos declarar variáveis em Java. Abreviações devem ser usadas somente se forem muito conhecidas no domínio do negócio. Por exemplo, você **evitar**:

```
int qtAlu;  
int quantAlunNt10;  
int nAlunosAprov;  
int totAlunosRep;
```

Essas regras não servem apenas para a linguagem Java, mas existe uma cultura muito forte entre os bons programadores Java que prezam pela clareza do código, e um nome de variável mal definido pode atrapalhar muito a legibilidade do código.

7.12. Operadores aritméticos

Existem 5 operadores aritméticos em Java que podemos usar para efetuar cálculos matemáticos. Uma operação pode ser de adição (+), subtração (-), multiplicação (*), divisão (/) ou módulo (%). Só para lembrar quem faltou nas aulas de matemática no colégio, módulo é o resto da divisão entre dois números. Outras operações como exponenciação (potência), raiz quadrada e outras são fornecidas de maneiras diferentes (que estudaremos mais adiante).

Vejamos um exemplo simples usando as 5 operações aritméticas:

```
int soma = 2 + 10;  
int subtracao = 6 - 10;  
int multiplicacao = 8 * 3;  
int divisao = 8 / 2;  
int resto = 7 % 2;  
  
System.out.println(soma);  
System.out.println(subtracao);  
System.out.println(multiplicacao);  
System.out.println(divisao);  
System.out.println(resto);
```

Os resultados das operações acima são: 12, -4, 24, 4 e 1.

No último exemplo, calculamos valores literais (digitados “na mão”). Podemos ainda calcular valores de variáveis, tornando o programa muito mais dinâmico.

```
int notaAluno1 = 99;  
int notaAluno2 = 80;  
int notaAluno3 = 53;  
  
int totalGeral = notaAluno1 + notaAluno2 + notaAluno3;  
System.out.println(totalGeral);
```

Pense rápido! Qual será o resultado da operação acima?

Puxa vida... você pensou rápido. O resultado é exatamente 232.

Para praticar um pouco mais, queremos agora descobrir qual é a média de notas dos 3 alunos que temos. O que você acha que acontece se dividirmos por 3?

```
int totalGeral = notaAluno1 + notaAluno2 + notaAluno3 / 3;
```

A idéia é muito boa. Para descobrirmos a média de notas de 3 alunos, basta somarmos todas as notas e dividir por 3, mas como fizemos no exemplo acima, estamos dividindo a última nota por 3, e não o resultado da somatória de todas as notas. Por isso, o resultado dessa operação é 196.

Para ficar correto, temos que agrupar a somatória usando parênteses, assim, dizemos ao compilador que queremos realizar a operação de soma antes da divisão.

```
int totalGeral = (notaAluno1 + notaAluno2 + notaAluno3) / 3;
```

Agora sim, o resultado da operação ficará correto, resultando na média de 77 pontos por aluno. Nada mal.

7.13. Tipos primitivos

Nos exemplos anteriores, vimos como usar variáveis para armazenar apenas valores inteiros. Agora vamos estudar como criar variáveis para armazenar valores do tipo ponto-flutuante (com casas decimais). Por exemplo, vamos declarar uma variável e atribuir um valor com o preço de um produto:

```
double precoProduto = 9.43;
```

O tipo `double` é capaz de armazenar valores reais, que possuem casas decimais, mas também pode armazenar números inteiros.

Outro tipo primitivo bastante usado é o `boolean`. O tipo booleano pode armazenar os valores verdadeiro ou falso (`true` ou `false`). Por exemplo:

```
boolean alunoMatriculado = true; // recebe valor "verdadeiro"  
boolean clienteBloqueado = false; // recebe valor "falso"
```

Os valores literais `true` e `false` são palavras reservadas do Java. Não é válido atribuir números a uma variável booleana. Não existe nenhuma referência entre um valor booleano e os números 0 e 1 ou qualquer outro número, como em outras linguagens que talvez você já conheça, por isso, o código abaixo é inválido e não compila:

```
boolean alunoMatriculado = 1; // não compila
boolean clienteBloqueado = 0; // não compila
```

Uma variável booleana pode receber como valor uma expressão com operadores de comparação ou igualdade, mas estudaremos isso em breve.

Depois de aprender como usar tipos inteiros, reais e booleanos, vamos conhecer mais sobre outros tipos primitivos do Java.

Os tipos de dados básicos (tipos primitivos) da linguagem Java são: `boolean`, `char`, `byte`, `short`, `int`, `long`, `float` e `double`.

A tabela abaixo mostra os tipos primitivos e a capacidade de armazenamento de cada um.

Tipo	Tamanho (bits)	Menor valor	Maior valor
<code>boolean</code>	1	<code>false</code>	<code>true</code>
<code>char</code>	16	0	$2^{16} - 1$
<code>byte</code>	8	-2^7	$2^7 - 1$
<code>short</code>	16	-2^{15}	$2^{15} - 1$
<code>int</code>	32	-2^{31}	$2^{31} - 1$
<code>long</code>	64	-2^{63}	$2^{63} - 1$
<code>float</code>	32	-	-
<code>double</code>	64	-	-

Como pode ver na tabela dos tipos primitivos do Java, o tipo `boolean` ocupa apenas 1-bit para armazenar um valor booleano, ou seja, este tipo precisa apenas de uma da menor unidade de armazenamento na computação.

Uma variável do tipo `char` ocupa 16-bit, ou seja, 2 bytes (cada byte tem 8 bits) para armazenar um valor que representa um caractere. Veja abaixo alguns exemplos de declaração e atribuição de variáveis do tipo `char`:

```
char turmaAluno1 = 'A';
char tipoCliente = '2';
char simbolo = '@';
System.out.println(turmaAluno1);
System.out.println(tipoCliente);
System.out.println(simbolo);
```

O tipo `char` não pode ser usado para armazenar um texto. Na verdade, este tipo é capaz de armazenar apenas um caractere. Nós aprenderemos o tipo `String` ainda neste curso para armazenar textos (com mais de 1 caractere, naturalmente).

Os tipos `byte` e `short` são tipos numéricos inteiros com uma capacidade de armazenamento menor que o tipo `int`. Para saber exatamente o menor e maior número que

cada tipo suporta, basta fazer o cálculo da potência, conforme apresentado na tabela. Por exemplo, o menor valor do tipo `byte` é -2^7 , ou seja, -128 , e o maior é $2^7 - 1$, ou seja, 127 .

O tipo `long` é um tipo numérico inteiro (longo) com capacidade de armazenamento superior ao tipo `int`. Para se ter uma idéia, o tipo `long` é capaz de armazenar o valor máximo igual a 9223372036854775807 , enquanto o `int` suporta até o número 2147483647 .

No exemplo abaixo, declaramos e atribuímos uma variável para armazenar o número total de habitantes na cidade de Uberlândia/MG. Esta variável poderia ser do tipo `int`, mas já que estamos falando de `long`, exageramos um pouco no exemplo e declaramos como um inteiro longo.

```
long populacaoUberlandia = 650000;  
System.out.println(populacaoUberlandia);
```

O código acima funciona, não temos nenhum problema com ele. Agora veja o código abaixo:

```
long populacaoMundial = 70000000000; // não compila  
System.out.println(populacaoMundial);
```

O último exemplo não compila! Pode parecer estranho, mas isso acontece porque o número 70000000000 não é compatível com o tipo `int`. Espere... mas não estávamos usando um tipo `int`, e sim um `long`. Certo?... Sim e não, certo e errado! Quando atribuímos um valor literal (digitado manualmente no código-fonte) a uma variável do tipo `long`, o valor literal é por natureza do tipo `int`, ou seja, os literais numéricos inteiros são do tipo `int` por padrão, a não ser se usarmos um pequeno truque. Veja como é fácil:

```
long populacaoMundial = 70000000000L; // compila!  
System.out.println(populacaoMundial);
```

A única diferença do último exemplo para o que não compila é a letra `L` após o número 70000000000 . Ao incluir a letra `L` ao final de um número literal, indicamos ao compilador que queremos que o número seja interpretado como um tipo `long`, e não um `int`. Capcioso, não?

Os tipos `float` e `double` são os únicos tipos ponto-flutuante, que são capazes de armazenar números reais (com casas decimais). Enquanto o tipo `float` possui 32-bit de precisão, o `double` possui 64-bit.

Já vimos como declarar variáveis do tipo `double`, agora tentaremos fazer o mesmo com o tipo `float`.

```
float saldoConta = 1030.43; // não compila  
System.out.println(saldoConta);
```

O código acima não compila porque todo literal decimal em Java é por padrão do tipo `double` (independente do valor). Por isso, precisamos mais uma vez tirar uma “carta na

manga"! Para fazer o código compilar e rodar como desejamos, precisamos apenas incluir a letra F após o número 1030.43.

```
float saldoConta = 1030.43F; // compila!  
System.out.println(saldoConta);
```

A letra F diz ao compilador que queremos que o número seja entendido como um valor do tipo float, e não double.

Os tipos float e double não devem ser usados para armazenar valores que devem ser precisos, como valores monetários. Para isso, nós vamos aprender outro tipo (que não é primitivo) mais a frente chamado BigDecimal. Até lá, você pode usar float e double para o que precisar, e você não será penalizado por nós.

7.14. Outros operadores de atribuição

Você já sabe para que serve o operador de atribuição = (igual), pois já usamos em vários exemplos anteriores. Agora vamos estudar outros operadores de atribuição que combinam com operadores aritméticos. Para começar, vejamos um exemplo do operador +=, que atribui um valor à variável somando o valor da própria variável com o número (ou variável) à direita do operador.

```
int total = 10;  
total += 3;  
System.out.println(total);
```

O código acima imprime na tela o número 13. A linha total += 3 é o mesmo que total = total + 3. Então porque usar essa forma abreviada? A resposta é óbvia... simplesmente para abreviar.

Outros operadores de atribuição básicos da linguagem Java são -=, *=, /= e %=. Veja um exemplo usando todos eles.

```
int total = 10;  
total += 3; // soma total com 3  
System.out.println(total);  
total -= 1; // subtrai total com 1  
System.out.println(total);  
total *= 2; // multiplica total por 2  
System.out.println(total);  
total /= 4; // divide total por 4  
System.out.println(total);  
total %= 5; // resto de total dividido por 5  
System.out.println(total);
```

Existem outros operadores de atribuição em Java para manipulação de bits, o que não é nosso foco neste curso.

7.15. Conversão de tipos primitivos

Durante a programação de um sistema, pode surgir a necessidade de atribuir a uma variável o valor de outra, porém de tipos diferentes. Isso é chamado de *casting* (conversão), pois antes da atribuição, um processo de conversão de um tipo para o outro deve ser realizado pela JVM. Vejamos o seguinte exemplo:

```
// declaramos a variável x do tipo long
long x = 10;

// agora tentamos atribuir x a y, do tipo int
int y = x; // não compila
```

O código acima não compila, pois não é possível atribuir `x` à `y`. Pode parecer estranho, pois `x` possui o valor `10`, que é perfeitamente compatível com o tipo `int` (lembre-se que `int` possui um limite máximo até o número `2147483647`). Porque então não compila?

Não tem como o compilador saber que a variável `x` possui o valor `10`, pois como o próprio nome diz, é uma variável! Quem garante que `x` não possa ter um valor absurdamente longo, que não cabe em um `int`? Por isso, para garantir que nada vai dar de errado em tempo de execução de seu programa, o compilador Java nos ajuda evitando que nosso programa seja compilado.

Para clarear um pouco mais, imagine o seguinte: um tipo `long` possui capacidade de 64 bits. É como se existissem 64 pequenos espaços na memória do computador para armazenar um valor. Mesmo que o número seja bastante pequeno, todos os espaços sempre serão ocupados.

Tamanho do tipo long



Já um tipo `int` possui a capacidade de 32 bits. Usando a mesma analogia, são como 32 pequenos espaços na memória para armazenar um valor.

Tamanho do tipo int



Agora imagine que você não saiba qual valor possui dentro das variáveis (o compilador não sabe) e responda: você consegue colocar com segurança o valor que tem na variável do tipo `long` dentro da variável do tipo `int`?

Se você entendeu bem e analisou os fatos friamente, com certeza respondeu que “não”. É impossível saber com certeza se o valor do tipo `long` caberá na variável do tipo `int`.

Mas Java é uma linguagem muito poderosa, e por isso não nos deixaria na mão. Se você tiver certeza ou precisar muito fazer essa conversão milagrosa, poderá forçar um *casting*, ou seja, você pode dizer ao compilador Java: “tudo bem, eu tenho mais que 18 anos e assumo todos os riscos que isso possa trazer”. Veja como é fácil:

```
long x = 10;

// agora compila, mas os riscos são todos seus
int y = (int) x;
System.out.println(y);
```

Graças à instrução de *casting* que incluímos, a variável `x` será convertida para `int` e atribuída à `y`. Neste caso, não existirá nenhum efeito colateral, pois o código é bastante simples e estamos vendo que o valor de `x` é realmente 10, porém no mundo real muitas vezes essa não é a situação, portanto, muito cuidado nessa hora.

E se ignorarmos as forças maiores e solicitarmos um *casting* de uma variável do tipo `long` tão longa que não cabe em um `int`? Um exemplo diz mais que mil palavras. Vejamos:

```
long x = 9300000035L;

// o valor mudará absurdamente (cuidado)
int y = (int) x;
System.out.println(y);
```

A execução do código acima imprimirá na tela o número 710065443. O que tem haver esse número com 9300000035? Nada... é verdade, mas assumimos todos os riscos, forçamos o *casting* da variável `x` e aconteceu o pior, o valor da variável `x` não coube em um tipo `int`, e por isso perdemos o valor original e ganhamos um outro que não queríamos.

Neste caso, o correto a fazer seria não fazer! Isso mesmo, o correto seria não fazer o *casting*, pois como bons programadores nós deveríamos saber que a variável `x` poderia em algum momento ter um valor tão longo ao ponto de não ser possível fazer a conversão com segurança.

Mas porque o valor mudou para outro totalmente diferente? Porque estávamos usando muitos dos 64 “espaços” (bits) do tipo `long`, e a conversão eliminou vários destes “espaços” para fazer caber dentro do tipo `int`. Quando fazemos isso, estamos perdendo bits que representam o número original, e por isso é natural que o resultado seja outro bem diferente.

Outra situação bastante comum que necessitamos fazer *casting* é de variáveis do tipo `int` para `long`. Veja um exemplo:

```
int y = 102344;

long x = y; // casting feito automaticamente
System.out.println(x);
```

O código acima compila e executa sem problemas. Quando tivermos um tipo `int` e desejarmos atribuir a outra variável do tipo `long`, podemos fazer isso sem correr riscos, por isso, não é necessário instruir ao compilador que você quer que seja feita uma conversão (ele fará automaticamente).

As conversões de tipos `float` para `double` e vice-versa acontecem da mesma forma que para os tipos `int` e `long`. Veja um exemplo:

```
double a = 20.5;

// você assume os riscos desta conversão
float b = (float) a;
System.out.println(b);

float c = 934.5f;

// conversão automática
double d = c;
System.out.println(d);
```

Como o tipo `double` possui 64 bits, é necessário deixar explícito a conversão para `float`, pois existem riscos de perder informação, já o contrário (de `float` para `double`) você não precisa dizer que existirá uma conversão, pois o Java fará isso de forma implícita para você.

Outra situação comum acontece quando precisamos atribuir valores de tipos ponto-flutuante para variáveis inteiras. O exemplo abaixo é uma tentativa de fazer isso:

```
double largura = 100;
int tamanho = largura; // não compila
```

O último exemplo não compila, pois o compilador não assume o risco de converter o valor de uma variável do tipo `double` (ou `float`) para uma variável do tipo `int`, `long` ou qualquer outra inteira.

Mesmo sabendo que o conteúdo da variável `largura` seria perfeitamente representada dentro da variável `tamanho`, o coitado do compilador não consegue identificar isso. Se quisermos realmente assumir todos os riscos, precisamos deixar explícita a conversão:

```
double largura = 100;
int tamanho = (int) largura; // agora sim, compila!
System.out.println(tamanho);
```

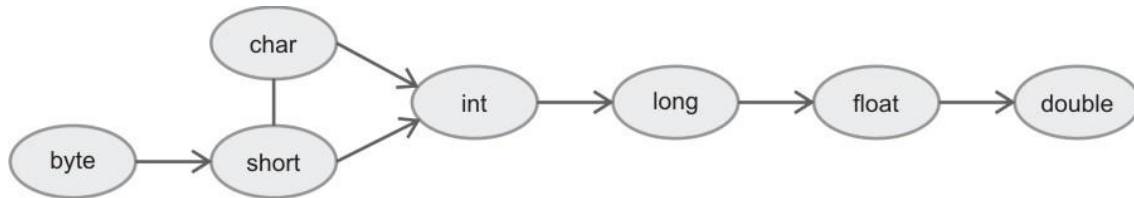
No caso do exemplo acima, está visível que não haverá nenhum problema. Mas e o código abaixo? O que acontecerá?

```
double largura = 100.37;
int tamanho = (int) largura; // compila, mas perdemos precisão
System.out.println(tamanho);
```

A execução do exemplo acima imprimirá na tela o número `100`. Isso acontece porque fizemos a conversão de `double` para `int`, mas dessa vez perdemos precisão. O número

100.37 não pôde ser representado como um inteiro, e por isso ele foi “truncado”, ou seja, o valor decimal foi eliminado sem dó nem piedade.

Se você precisar fazer *casting* de tipos diferentes ao que estudamos, veja a imagem abaixo e siga algumas dicas.



- No sentido das flechas, a conversão é implícita, ou seja, aquela que você não precisa escrever nada (e mesmo assim acontecerá automaticamente).
- No sentido contrário ao das flechas, a conversão deve ser explícita, ou seja, você deve escrever no código-fonte que deseja assumir os riscos.

7.16. Promoção aritmética

Quando calculamos duas variáveis ou valores do tipo `int`, por exemplo, o resultado é do tipo `int`.

```
int x = 10;
int y = 5;
int z = x + y;
System.out.println(z);
```

Se realizarmos um cálculo com duas variáveis ou valores do tipo `long`, o resultado é do tipo `long`.

```
long x = 10;
long y = 5;
long z = x * y;
System.out.println(z);
```

Agora, se realizarmos uma operação entre uma variável do tipo `int` com uma variável do tipo `long`, podemos atribuir o resultado em um tipo `int`?

```
int x = 10;
long y = 5;
int z = x * y; // não compila!
System.out.println(z);
```


Não, não... não podemos! Quando realizamos operações aritméticas entre variáveis de tipos diferentes, o resultado será igual ao maior tipo, e isso se chama **promoção aritmética**. Como `long` tem maior capacidade de armazenamento que `int`, o resultado é do tipo `long`, por isso, seria correto atribuímos em uma variável desse tipo.

```
int x = 10;
long y = 5;
long z = x * y; // agora compila, pois o resultado é long
System.out.println(z);
```

Para sabermos exatamente qual o tipo resultante em uma operação aritmética, basta olharmos a tabela de tipos primitivos estudada anteriormente. O tipo com a maior capacidade em bits será o tipo do resultado do cálculo.

A exceção a essa regra é quando realizamos uma operação entre números de tipos inteiros (`int`, `long`, etc) e ponto-flutuante (`float` ou `double`).

```
long x = 10;
float y = 9.34;
long z = x * y; // não compila
System.out.println(z);
```

O exemplo acima não compila, pois o resultado de uma operação entre `float` e `long` é `float`. Apesar de `long` ter 64 bits e `float` ter 32 bits, `float` leva a vantagem por ser ponto-flutuante, portanto, o correto seria atribuir o resultado dessa operação em um tipo `float`.

```
long x = 10;
float y = 9.34;
float z = x * y; // compila
System.out.println(z);
```

Por último, apenas para você refletir um pouco, tente descobrir o que será impresso na tela com o código abaixo:

```
int x = 3;
int y = 2;
float z = x / y;
System.out.println(z);
```

O exemplo é bem simples, estamos simplesmente tentando dividir `x` por `y`, ou seja, 3 por 2. A grande questão é que estamos dividindo dois números inteiros, portanto, o resultado também será do tipo inteiro.

Se você acha que aparecerá na tela o valor 1.5 (3 dividido por 2 é 1.5), se enganou! Nós veremos 1.0 ao executar esse código. Apesar de atribuímos o resultado em um tipo `float` (que suporta casas decimais), o cálculo é feito usando duas variáveis do tipo `int`, portanto, o resultado também é do tipo `int`.

Se não quisermos perder os valores decimais do resultado da operação, podemos ter pelo menos uma das variáveis que participam do cálculo do tipo `float`.

```
int x = 3;
float y = 2;
float z = x / y;
System.out.println(z);
```

Agora sim, o resultado que veremos na tela será 1.5, pois `x` é promovido a `float` no momento da operação.

Se não existir essa possibilidade (de mudar o tipo da variável para `float`), podemos fazer um *casting* de `x` ou `y` para `float` antes de realizar o cálculo.

```
int x = 3;
int y = 2;
float z = x / (float) y; // o casting acontece antes do cálculo
System.out.println(z);
```

O resultado continua sendo 1.5, pois convertermos o valor de `y` para `float`, obrigando `x` a ser promovido também para `float` durante a operação aritmética.

7.17. Trabalhando com strings

Para apresentarmos um texto na tela usando Java, tudo que precisamos fazer é colocá-lo entre aspas duplas dentro de um `System.out.println`.

```
System.out.println("Oi galera do bem");
```

Se precisarmos juntar um texto com o valor de uma variável, podemos concatená-los. Na linguagem Java, usamos o símbolo `+` (mais) para fazer isso.

```
int x = 10;
int y = 5;
int z = x + y; // adição
System.out.println("Resultado: " + z); // concatenação
```

Quando usamos o `+` em textos, ele é entendido pelo compilador como concatenação, e não adição.

No exemplo acima, poderíamos eliminar a variável `z` e efetuar o cálculo diretamente no `println`. Veja abaixo uma tentativa de fazer isso:

```
int x = 10;
int y = 5;
System.out.println("Resultado: " + x + y);
```

A execução do último exemplo exibirá na tela “Resultado: 105”, pois o valor da variável `x` é concatenado com o valor da variável `y`. Isso acontece porque o compilador verifica que existe um texto sendo concatenado com a variável `x`, e faz o mesmo para a variável `y`.

Agora veja outro exemplo:

```
int x = 10;
int y = 5;
System.out.println(x + y + " foi o resultado");
```

Você apostaria que o código acima imprime na tela “15 foi o resultado”?

Apesar de parecer estranho, é isso que acontece. Neste caso, as variáveis `x` e `y` são somadas, e não concatenadas. Isso acontece porque o compilador só começa a concatenar a partir do momento que encontra um texto.

E se quisermos efetuar o cálculo de `x` com `y` depois que um texto foi digitado, como na primeira situação? Basta incluirmos a operação entre parênteses para dizer ao compilador que a operação tem precedência.

```
int x = 10;
int y = 5;
System.out.println("Resultado: " + (x + y));
```

Agora o resultado será “Resultado: 15”.

Para declarar variáveis que contenham textos, usamos o tipo `String`.

```
String nome = "Maria";
```

Apesar de o tipo `String` ser nativo da linguagem Java, ele não é um tipo primitivo. Por enquanto você não precisa se preocupar com isso. Nós aprofundaremos mais sobre o tipo `String` mais adiante.

No exemplo abaixo, declaramos a variável `nome` do tipo `String` e `idade` do tipo `int`, e depois imprimimos na tela uma mensagem concatenando o `nome` e a `idade` mais outros textos para formar uma frase completa.

```
String nome = "Maria";
int idade = 30;
System.out.println(nome + " tem " + idade + " anos");
```

O resultado na tela é “Maria tem 30 anos”.

7.18. Recebendo entrada de dados

Até agora, criamos exemplos usando variáveis com valores definidos diretamente no código-fonte (literals), sem entrada de dados pelo usuário. Está na hora de evoluir nossos códigos e aprender como solicitar informações ao usuário durante a execução do programa.

A forma mais simples de fazer isso em Java é usando uma classe chamada `Scanner`. Por enquanto, não se preocupe com o que é uma classe e nem o que é essa tal de `Scanner`. Concentre-se apenas em como deve ser feito para obter o resultado esperado e tenha paciência, pois você aprenderá tudo sobre classes nos capítulos sobre Orientação a Objetos.

```
import java.util.Scanner;

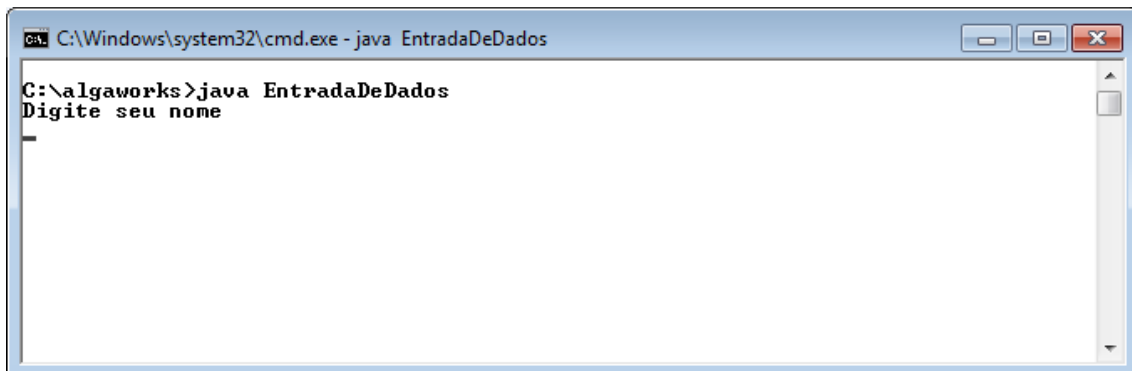
public class EntradaDeDados {

    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);

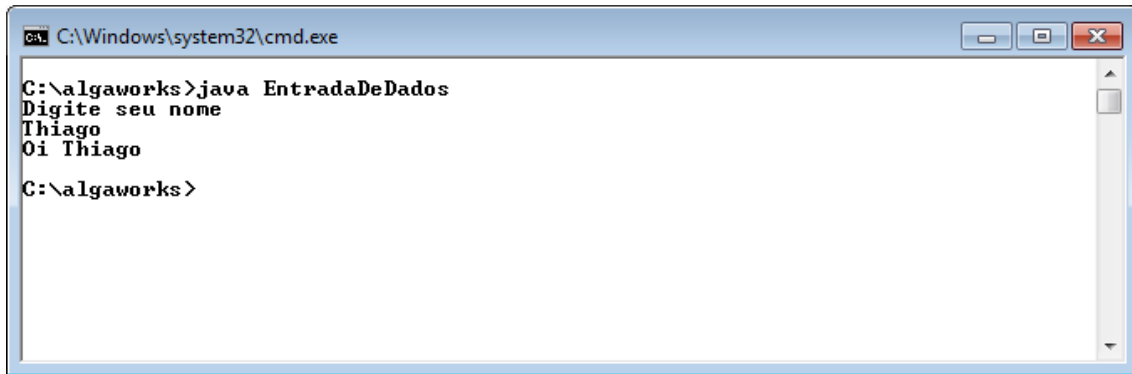
        System.out.println("Digite seu nome");
        String nome = entrada.nextLine();

        System.out.println("Oi " + nome);
    }
}
```

O exemplo acima solicita a entrada de dados e aguarda uma resposta do usuário.



Ao digitar alguma informação (no caso, o seu nome), o valor digitado é atribuído à variável `nome` e depois impresso uma mensagem com o nome concatenado.



```
C:\Windows\system32\cmd.exe
C:\algaworks>java EntradaDeDados
Digite seu nome
Thiago
Oi Thiago
C:\algaworks>
```

Para usar a classe `Scanner`, temos que importá-la em nosso código-fonte. Para fazer isso, basta incluir a linha abaixo no topo do arquivo do código-fonte.

```
import java.util.Scanner;
```

Novamente, não há motivos para se preocupar, pois você aprenderá sobre importações mais adiante. Apenas faça isso!

Declaramos uma variável nomeada como `entrada` do tipo `Scanner`. Esse tipo não é um tipo primitivo, mas não precisa nem dizer que você aprenderá sobre isso mais pra frente, certo?

```
Scanner entrada = new Scanner(System.in);
```

Finalmente, para solicitar a entrada de um texto, invocamos o método `nextLine()` através da variável `entrada`.

```
String nome = entrada.nextLine();
```

Neste momento, o programa ficará paralisado aguardando o usuário digitar alguma coisa. Quando a tecla *Enter* for pressionada, o conteúdo informado será atribuído à variável `nome`.

A classe `Scanner` possui vários métodos para obter dados já convertidos em tipos específicos. No segundo exemplo, usamos dois novos métodos para obter entrada de dados dos tipos `int` e `double`. O programa solicitará, além do nome do usuário, o peso e altura e fará o cálculo do Índice de Massa Corporal (IMC).

```
Scanner entrada = new Scanner(System.in);
```

```
System.out.print("Nome: ");
```

```
// obtém entrada do tipo String
String nome = entrada.nextLine();
```

```
System.out.print("Peso: ");

// obtém entrada do tipo int
int peso = entrada.nextInt();

System.out.print("Altura: ");

// obtém entrada do tipo double
double altura = entrada.nextDouble();

double imc = peso / (altura * altura);

System.out.println("IMC de " + nome + ": " + imc);
```

Usamos `System.out.print` (e não `println`) para exibir as mensagens que solicitam dados ao usuário. A diferença de `print` com `println` é que o primeiro não exibe quebra de linha. Dessa forma, o usuário pode digitar os dados na frente da mensagem.



```
C:\Windows\system32\cmd.exe

C:\algaworks>java IndiceMassaCorporal
Nome: Sebastiao
Peso: 75
Altura: 1.82
IMC de Sebastiao: 22.6421929718633
C:\algaworks>
```

Não tente chamar o método `nextLine()` depois de ter chamado `nextInt()` ou `nextDouble()` através da mesma variável do `Scanner`, pois isso não funciona bem. Se precisar, crie duas variáveis do tipo `Scanner` para solucionar esse problema.

7.19. Operadores de comparação e igualdade

Operadores de comparação e de igualdade são usados para realizar comparações de conteúdos de variáveis ou valores literais. Os resultados das comparações sempre resultam em valores booleanos.

Os operadores de comparação disponíveis são `>` (maior), `>=` (maior ou igual), `<` (menor) e `<=` (menor ou igual), e os operadores de igualdade são `==` (igual) e `!=` (diferente). O exemplo abaixo usa todos estes operadores.

```
boolean maior = b > a; // 'b' é maior que 'a'?
boolean maiorOuIgual = b >= a; // 'b' é maior ou igual a 'a'?
boolean menor = a < b; // 'a' é menor que 'b'?
boolean menorOuIgual = a <= 10; // 'a' é menor ou igual a '10'?
boolean igual = a == b - c; // 'a' é igual a 'b' menos 'c'?
boolean diferente = a != c; // 'a' é diferente de 'c'?
```

```
System.out.println(maior);  
System.out.println(maiorOuIgual);  
System.out.println(menor);  
System.out.println(menorOuIgual);  
System.out.println(igual);  
System.out.println(diferente);
```

Coincidentemente, todas as comparações acima resultam no valor booleano `true` (verdadeiro). Perceba que na quarta linha incluímos um valor literal para fazer a comparação, e na quinta linha fizemos uma operação aritmética subtraindo duas variáveis antes de realizar a comparação. Fizemos isso só para exercitarmos um pouco mais a linguagem Java.

Já que falamos de operadores de igualdade, vale a pena falarmos do operador unário ! (ponto de exclamação). Este operador serve para negar um valor booleano ou uma expressão booleana.

Se tivermos uma variável booleana com `true` atribuído a ela, podemos negar a própria variável para o valor passar a valer `false`.

```
boolean bloqueado = true;  
bloqueado = !bloqueado; // passa a valer false
```

Podemos também negar uma expressão que possui um operador de comparação. Por exemplo:

```
boolean resultado = !(b > a); // ruim, mas válido
```

O código acima compara se `b` é maior que `a` e inverte o resultado. É o mesmo que fazer `b <= a`, porém com uma legibilidade um pouco pior (pois exige mais raciocínio para entender o código).

7.20. Estruturas de controle `if`, `else if` e `else`

Qualquer bom programa deve ser capaz de tomar decisão durante sua execução, seguindo caminhos diferentes de acordo a lógica do sistema.

Como em muitas linguagens de programação, em Java, usamos a instrução `if` (se) para controlar um fluxo básico.

No exemplo abaixo, melhoramos um pouco o código-fonte que calcula o IMC de uma pessoa para exibir uma mensagem caso seu peso esteja abaixo do ideal. Para saber se o peso está abaixo do ideal, precisamos verificar se o resultado do cálculo do IMC é menor que 18.5.

```
Scanner entrada = new Scanner(System.in);  
  
System.out.print("Nome: ");  
String nome = entrada.nextLine();  
  
System.out.print("Peso: ");  
int peso = entrada.nextInt();
```

```
System.out.print("Altura: ");  
double altura = entrada.nextDouble();  
  
double imc = peso / (altura * altura);  
  
if (imc < 18.5) {  
    System.out.println("Abaixo do peso ideal.");  
}
```

Veja que usamos uma instrução `if` para comparar se a variável `imc` é menor que 18.5, através da expressão `imc < 18.5`.

```
if (imc < 18.5) {  
    System.out.println("Abaixo do peso ideal.");  
}
```

A instrução `if` deve receber uma expressão booleana agrupada por parênteses.

No exemplo acima, a chamada de `System.out.println` só será executada caso o valor da variável `imc` seja menor que 18.5.

Se o cálculo do IMC não for menor que 18.5, queremos que uma nova condição seja verificada para identificar se o peso do usuário é o ideal. Para isso, usamos a instrução `else if` (senão se), que recebe uma expressão booleana e é analisada apenas se a primeira instrução `if` for falsa.

```
if (imc < 18.5) {  
    System.out.println("Abaixo do peso ideal.");  
} else if (imc < 25) {  
    System.out.println("Peso ideal.");  
}
```

Agora, caso o IMC do usuário não seja menor que 18.5, verificamos se é menor que 25 (ou seja, maior que 18.5 e menor que 25) e imprimimos outra mensagem dizendo que o peso é ideal.

Finalizamos nosso programa incluindo outras condições que indicam o grau de obesidade de uma pessoa. Veja no último caso que usamos a instrução `else` (caso contrário ou senão). Caso nenhuma das condições expressadas no `if` e nos `else ifs` forem verdadeiras, o bloco de código de `else` será executado.

```
if (imc < 18.5) {  
    System.out.println("Abaixo do peso ideal.");  
} else if (imc < 25) {  
    System.out.println("Peso ideal.");  
} else if (imc < 30) {  
    System.out.println("Acima do peso.");  
} else if (imc < 35) {  
    System.out.println("Obesidade grau I.");  
} else if (imc < 40) {  
    System.out.println("Obesidade grau II.");  
} else {  
    System.out.println("Obesidade grau III.");  
}
```



```
}
```

A instrução `else` não recebe nenhuma expressão booleana, pois ela será executada apenas se todas as instruções anteriores forem falsas.

7.21. Programar `ifs` sem abrir e fechar blocos é legal?

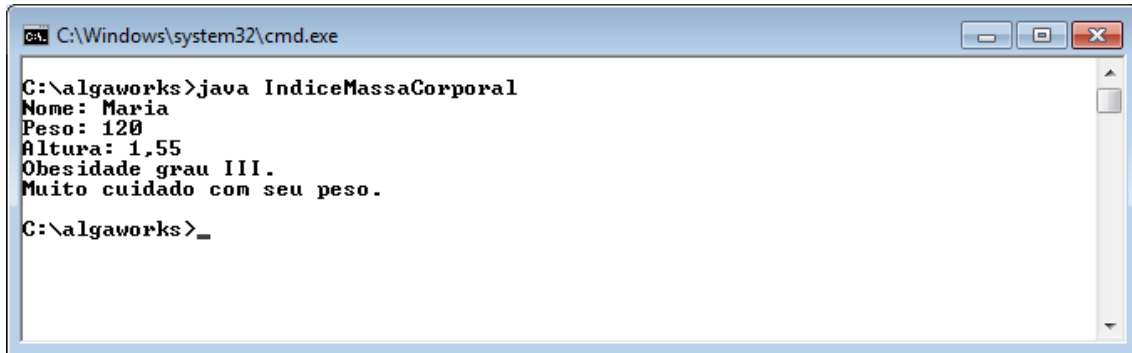
Existe a possibilidade modificar o código-fonte do último exemplo programado sem mudar o comportamento do programa, eliminando as aberturas e fechamentos de blocos (chaves).

```
if (imc < 18.5)
    System.out.println("Abaixo do peso ideal.");
else if (imc < 25)
    System.out.println("Peso ideal.");
else if (imc < 30)
    System.out.println("Acima do peso.");
else if (imc < 35)
    System.out.println("Obesidade grau I.");
else if (imc < 40)
    System.out.println("Obesidade grau II.");
else
    System.out.println("Obesidade grau III.");
```

O resultado é o mesmo, e, apesar de parecer que o código está mais limpo e bonito, existe um risco muito grande quando programamos assim. Imagina se precisamos adicionar uma nova mensagem a ser exibida para o usuário quando o grau de obesidade for III, poderíamos tentar incluir uma linha que deve ser executada na instrução `else`.

```
if (imc < 18.5)
    System.out.println("Abaixo do peso ideal.");
else if (imc < 25)
    System.out.println("Peso ideal.");
else if (imc < 30)
    System.out.println("Acima do peso.");
else if (imc < 35)
    System.out.println("Obesidade grau I.");
else if (imc < 40)
    System.out.println("Obesidade grau II.");
else
    System.out.println("Obesidade grau III.");
    System.out.println("Muito cuidado com seu peso.");
```

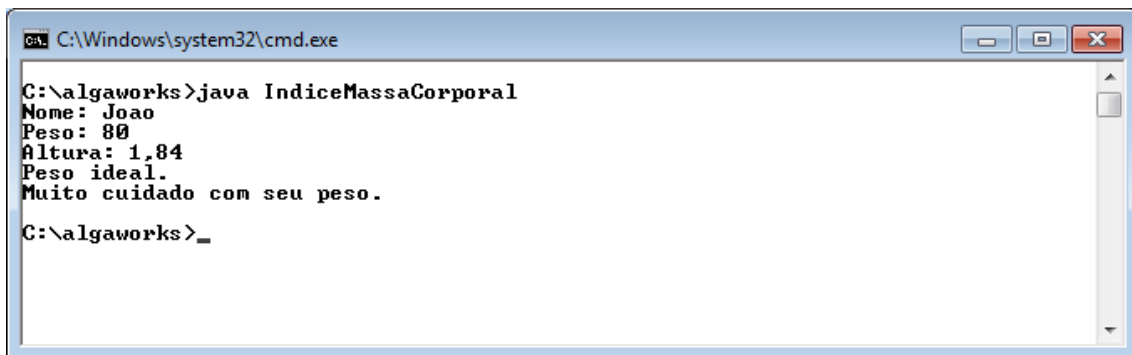
Ao executar esse programa, se o IMC for acima de 40 (nível máximo de obesidade), as mensagens aparecerão como esperado, como se o nosso sistema estivesse funcionando corretamente.



```
C:\Windows\system32\cmd.exe

C:\algaworks>java IndiceMassaCorporal
Nome: Maria
Peso: 120
Altura: 1,55
Obesidade grau III.
Muito cuidado com seu peso.
C:\algaworks>_
```

Agora se executamos o programa novamente (sem fazer qualquer alteração) e informarmos um peso ideal para uma determinada altura, veja que algo de errado acontece.



```
C:\Windows\system32\cmd.exe

C:\algaworks>java IndiceMassaCorporal
Nome: Joao
Peso: 80
Altura: 1,84
Peso ideal.
Muito cuidado com seu peso.
C:\algaworks>_
```

A mensagem “Muito cuidado com seu peso” apareceu para alguém que tem peso ideal, e essa não era nossa intenção.

O problema é que, quando não usamos as chaves para abrir e fechar blocos, nos enganamos facilmente. Sem as chaves, cada instrução `if`, `else if` ou `else` faz referência apenas para a primeira instrução logo em seguida.

O último exemplo é como se tivéssemos feito:

```
...
else
    System.out.println("Obesidade grau III.");
System.out.println("Muito cuidado com seu peso.");
```

Ou ainda:

```
...
else {
    System.out.println("Obesidade grau III.");
}

System.out.println("Muito cuidado com seu peso.");
```

Por existir essa “facilidade” de errar, é recomendado que você evite programar `if/else if/else` sem usar blocos.

7.22. Escopo de variáveis

O escopo de variáveis define em qual parte do programa a variável pode ser referenciada através de seu nome.

Para usarmos uma variável, ela deve ser declarada antes, mas só isso não é suficiente. Deve-se observar se a variável não foi declarada em um bloco mais interno que o código que está tentando usá-la. O código-fonte abaixo é um exemplo que nem mesmo compila:

```
Scanner entrada = new Scanner(System.in);

System.out.print("Idade: ");
int idade = entrada.nextInt();
boolean podeDirigir = idade >= 18;

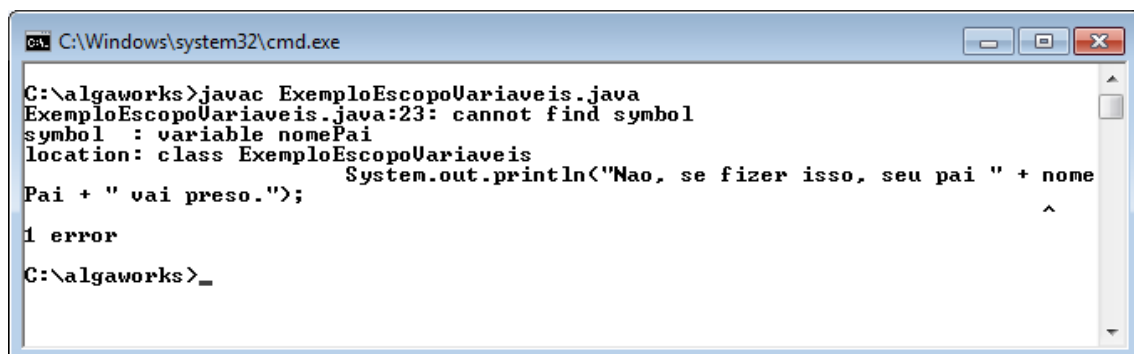
if (!podeDirigir) {
    System.out.print("Nome do pai: ");

    // variável nomePai está sendo declarada apenas para o
    // o bloco deste if
    String nomePai = entrada.next();
}

System.out.println("Voce pode dirigir? ");

if (podeDirigir) {
    System.out.println("Sim, claro.");
} else {
    // não compila! variável nomePai não está acessível
    System.out.println("Nao, se fizer isso, seu pai "
        + nomePai + " vai preso.");
}
```

O código acima não compila porque a variável `nomePai` ficaria visível apenas para o bloco do `if (!podeDirigir) { ... }`.



```
C:\Windows\system32\cmd.exe

C:\algaworks>javac ExemploEscopoVariaveis.java
ExemploEscopoVariaveis.java:23: cannot find symbol
symbol : variable nomePai
location: class ExemploEscopoVariaveis
    System.out.println("Nao, se fizer isso, seu pai " + nome
Pai + " vai preso.");
1 error
C:\algaworks>_
```

Quando criamos variáveis em Java, elas ficam acessíveis apenas dentro do bloco o qual ela foi declarada e seus sub-blocos.

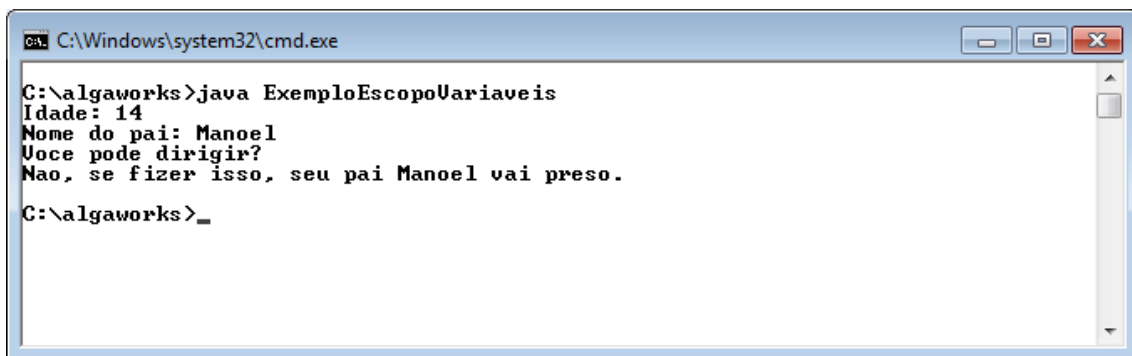
Para o exemplo anterior compilar, precisaríamos declarar a variável `nomePai` fora do bloco do `if`, deixando-a visível para todo o método `main` do programa.

```
String nomePai = "";

if (!podeDirigir) {
    System.out.print("Nome do pai: ");

    nomePai = entrada.next();
}
```

Agora o programa pode ser compilado e executado.



7.23. Operadores lógicos

Nos últimos exemplos, usamos as regras da Organização Mundial de Saúde para calcular o IMC. Existem outras regras mais detalhadas, como os da *NHANES II survey (USA 1976-1980)*³, que indicam aos seguintes critérios para adultos:

Condição	IMC em mulheres	IMC em homens
Abaixo do peso	Menor que 19.1	Menor que 20.7
No peso ideal	Entre 19.1 e 25.8	Entre 20.8 e 26.4
Um pouco acima do peso	Entre 25.9 e 27.3	Entre 26.5 e 27.8
Acima do peso ideal	Entre 27.4 e 32.3	Entre 27.9 e 31.1
Obeso	Maior que 32.3	Maior que 31.1

³ Overweight definition using body mass index. Steven B. Halls, MD. <http://www.halls.md/body-mass-index/overweight.htm>

Como você pôde ver, essas regras são um pouco mais complicadas, pois são diferentes entre homens e mulheres. Apesar disso, um programador experiente não deve sentir dificuldade para programá-las.

Existem diversas formas de codificar um programa que esses critérios. Uma forma seria aninhar (agrupar) `ifs`, colocando um bloco dentro de outro.

```
System.out.print("Nome: ");
String nome = entrada.nextLine();

System.out.print("Peso: ");
int peso = entrada.nextInt();

System.out.print("Altura: ");
double altura = entrada.nextDouble();

System.out.print("Sexo (1 para 'M' ou outro numero para 'F'): ");
char sexo = entrada.nextShort() == 1 ? 'M' : 'F';

double imc = peso / (altura * altura);

if (sexo == 'F') {
    if (imc < 19.1) {
        System.out.println("Abaixo do peso.");
    } else if (imc <= 25.8) {
        System.out.println("Peso ideal.");
    }
    // e continua...
} else {
    if (imc < 20.7) {
        System.out.println("Abaixo do peso.");
    } else if (imc <= 26.4) {
        System.out.println("Peso ideal.");
    }
    // e continua...
}
```

No exemplo acima, pedimos que usuário informe seu sexo, sendo que deve ser digitado o número 1 para que o sistema entenda que o sexo é masculino ou qualquer outro código se for feminino.

```
System.out.print("Sexo (1 para 'M' ou outro numero para 'F'): ");
char sexo = entrada.nextShort() == 1 ? 'M' : 'F';
```

Quando colocamos `ifs` dentro de `ifs`, estamos dizendo que o `if` mais interno só será avaliado caso o primeiro seja verdadeiro.

```
if (sexo == 'F') {

    // só será avaliado se sexo for Masculino
    if (imc < 19.1) {
        System.out.println("Abaixo do peso.");
    } else if (imc <= 25.8) {
```

```
        System.out.println("Peso ideal.");  
    }  
  
    // e continua...  
}
```

Antes que nos esqueçamos, estamos em um tópico sobre operadores lógicos! O último exemplo nada tem haver com isso, mas foi apenas uma motivação para falarmos agora sobre esses operadores.

Ao invés de incluirmos ifs dentro de ifs, podemos usar o operador lógico “E”, que é representado por &&.

```
if (sexo == 'F' && imc < 19.1) {  
    System.out.println("Abaixo do peso.");  
} else if (sexo == 'F' && imc <= 25.8) {  
    System.out.println("Peso ideal.");  
} else if (sexo == 'F' && imc <= 27.3) {  
    System.out.println("Um pouco acima do peso.");  
} else if (sexo == 'F' && imc <= 32.3) {  
    System.out.println("Acima do peso ideal.");  
} else if (sexo == 'F') {  
    System.out.println("Obeso.");  
} else if (sexo == 'M' && imc < 20.7) {  
    System.out.println("Abaixo do peso.");  
} else if (sexo == 'M' && imc <= 26.4) {  
    System.out.println("Peso ideal.");  
} else if (sexo == 'M' && imc <= 27.8) {  
    System.out.println("Um pouco acima do peso.");  
} else if (sexo == 'M' && imc <= 31.1) {  
    System.out.println("Acima do peso ideal.");  
} else if (sexo == 'M') {  
    System.out.println("Obeso.");  
}
```

O operador lógico && avalia as expressões do lado esquerdo e direito e retorna apenas um resultado booleano. Para a expressão completa ser verdadeira, tanto o lado direito como o lado esquerdo devem ser verdadeiras, mas para que a expressão completa seja falsa, pelo menos um lado deve ser falso.

Ainda existe o operador lógico “OU”, representado por ||. Podemos ainda mudar o código-fonte de nosso exemplo para usá-lo.

```
if ((sexo == 'F' && imc < 19.1) || (sexo == 'M' && imc < 20.7)) {  
    System.out.println("Abaixo do peso.");  
} else if ((sexo == 'F' && imc <= 25.8) || (sexo == 'M' && imc <= 26.4)) {  
    System.out.println("Peso ideal.");  
} else if ((sexo == 'F' && imc <= 27.3) || (sexo == 'M' && imc <= 27.8)) {  
    System.out.println("Um pouco acima do peso.");  
} else if ((sexo == 'F' && imc <= 32.3) || (sexo == 'M' && imc <= 31.1)) {  
    System.out.println("Acima do peso ideal.");  
} else {  
    System.out.println("Obeso.");  
}
```

Veja que usando o operador `||`, conseguimos diminuir bastante a quantidade de linhas de programação, porém deixamos as expressões booleanas dos `ifs` mais complexas, pois agrupamos duas expressões que usam o operador `&&` dentro de outra que usa o operador `||`.

O operador lógico `||` avalia as expressões dos dois lados e retorna um único resultado booleano. Para que a expressão completa seja verdadeira, pelo menos um lado deve ser verdadeiro.

Não existe uma regra geral que define o que é melhor, usar `ifs` dentro de `ifs` ou aumentar a complexidade das expressões booleanas. Normalmente, o que for mais simples de ser lido é o melhor.

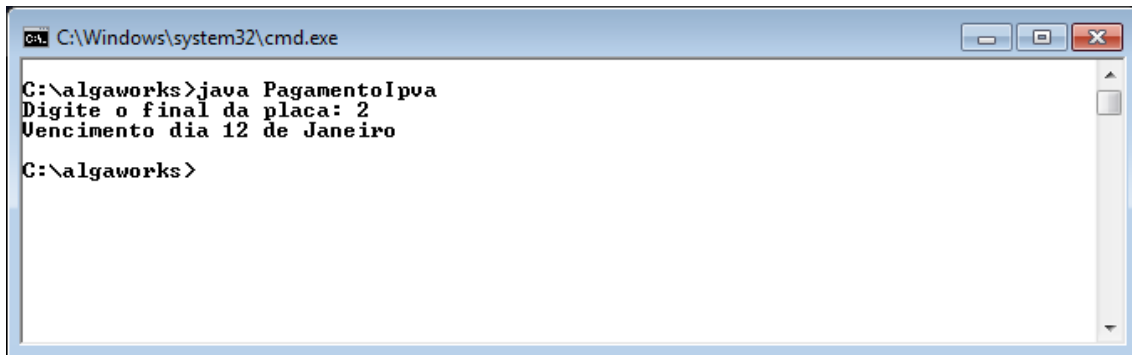
7.24. Estrutura de controle `switch`

A estrutura de controle `switch` recebe um valor inteiro (ou que possa ser convertido implicitamente para inteiro) e vários casos (possíveis caminhos de decisão). O primeiro caso que estiver de acordo com o valor passado para o `switch` inicia a execução das instruções do caso.

Vamos construir um programa que solicita o número final da placa de um veículo ao usuário e informa a data do vencimento do IPVA. Por enquanto, para simplificar, vamos programar apenas as opções de placas que terminam com os números “1” e “2”.

```
public class PagamentoIpva {  
  
    public static void main(String[] args) {  
        Scanner entrada = new Scanner(System.in);  
  
        System.out.print("Digite o final da placa: ");  
        int finalPlaca = entrada.nextInt();  
  
        switch (finalPlaca) {  
            case 1:  
                System.out.println("Vencimento dia 11 de Janeiro");  
            case 2:  
                System.out.println("Vencimento dia 12 de Janeiro");  
        }  
    }  
}
```

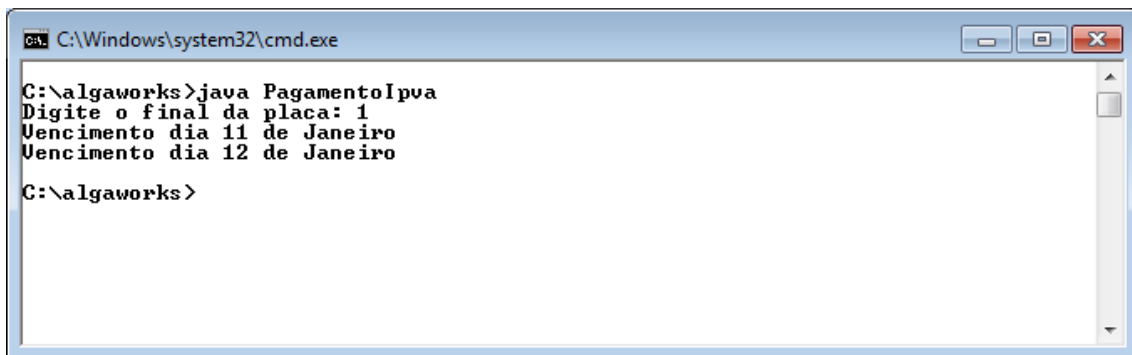
Se executarmos o exemplo acima e digitarmos o número “2”, visualizamos na tela a data de vencimento, como esperávamos.



```
C:\Windows\system32\cmd.exe

C:\algaworks>java PagamentoIpva
Digite o final da placa: 2
Vencimento dia 12 de Janeiro
C:\algaworks>
```

Agora vamos executar o programa novamente e informar o número “1”. Veja o resultado:



```
C:\Windows\system32\cmd.exe

C:\algaworks>java PagamentoIpva
Digite o final da placa: 1
Vencimento dia 11 de Janeiro
Vencimento dia 12 de Janeiro
C:\algaworks>
```

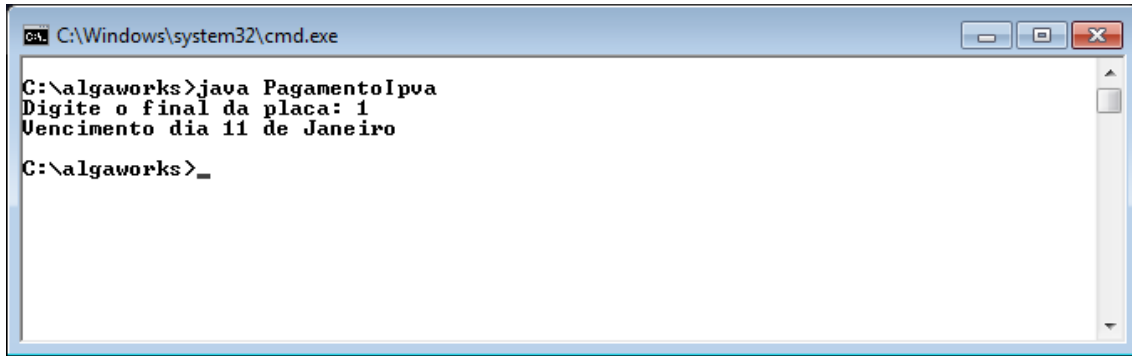
O programa executou as instruções do primeiro e segundo caso!

Isso não é um erro, mas um detalhe sobre o funcionamento do `switch`. Quando um caso encontra o valor igual ao passado para o `switch`, todas as instruções a partir desse caso são executadas, independente se os valores dos casos não têm nada haver com o valor do `switch`.

Se quisermos evitar isso, precisamos incluir a instrução `break` no fim de cada caso. Assim, a execução do `switch` é interrompida e as demais linhas de outros casos não são executadas.

```
switch (finalPlaca) {
    case 1:
        System.out.println("Vencimento dia 11 de Janeiro");
        break;
    case 2:
        System.out.println("Vencimento dia 12 de Janeiro");
        break;
}
```

Agora podemos compilar e executar o programa novamente.



```
C:\Windows\system32\cmd.exe

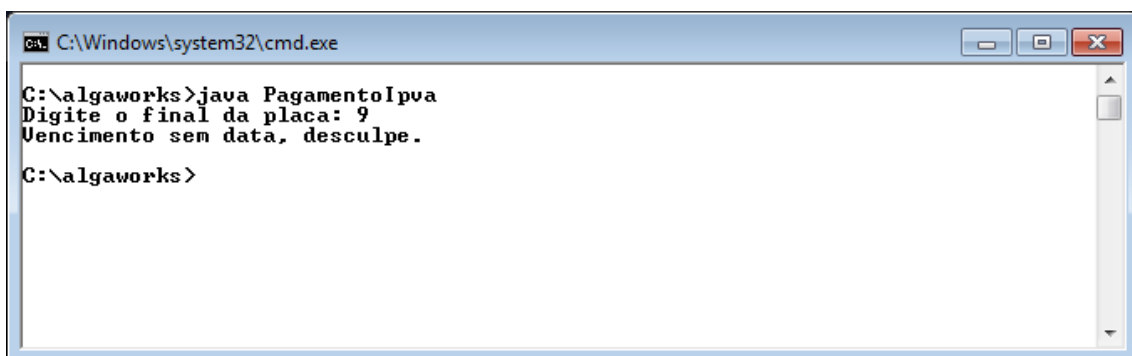
C:\algaworks>java PagamentoIpva
Digite o final da placa: 1
Vencimento dia 11 de Janeiro
C:\algaworks>
```

Nada acontece se informarmos um número que não existe um caso para ele (todos os casos são ignorados). Se precisarmos executar alguma coisa quando nenhum caso for encontrado, podemos usar o `default`.

O `default` é um bloco padrão, que é executado quando nenhum caso é satisfeito. Geralmente o colocamos no final do `switch`, mas pode ser incluído em qualquer ordem.

```
switch (finalPlaca) {
    case 1:
        System.out.println("Vencimento dia 11 de Janeiro");
        break;
    case 2:
        System.out.println("Vencimento dia 12 de Janeiro");
        break;
    default:
        System.out.println("Vencimento sem data, desculpe.");
}
```

Ao compilarmos e executarmos o programa e digitar um número que não satisfaz nenhum caso, por exemplo, “9”, veja que agora o caso `default` é processado.



```
C:\Windows\system32\cmd.exe

C:\algaworks>java PagamentoIpva
Digite o final da placa: 9
Vencimento sem data, desculpe.
C:\algaworks>
```

7.25. Operador ternário

O algoritmo abaixo solicita ao usuário a idade e exibe uma mensagem dizendo se a idade é classificada como adulto ou criança/adolescente. Para fazer isso, usamos apenas o que aprendemos até agora.

```
Scanner entrada = new Scanner(System.in);

System.out.print("Digite sua idade: ");
int idade = entrada.nextInt();

String indicacao = "";

if (idade >= 18) {
    indicacao = "adulto";
} else {
    indicacao = "criança/adolescente";
}

System.out.println("Resultado: " + indicacao);
```

O operador ternário é uma forma simples de reduzir linhas de código com instruções `if`. O operador permite que um valor seja atribuído a uma variável baseado em uma expressão booleana.

O exemplo abaixo faz o mesmo que o último código-fonte, porém usando o operador ternário:

```
Scanner entrada = new Scanner(System.in);

System.out.print("Digite sua idade: ");
int idade = entrada.nextInt();

String indicacao = (idade >= 18) ? "adulto" : "criança/adolescente";

System.out.println("Resultado: " + indicacao);
```

O operador ternário possui a seguinte sintaxe:

```
expressão booleana ? valor caso verdadeiro : valor caso falso;
```

Também é possível usar operadores ternários aninhados. O exemplo abaixo atribui à variável `indicacao` o valor “adulto”, “criança” ou “adolescente”, dependendo do valor da variável `idade`.

```
String indicacao =
    (idade >= 18) ? "adulto" : (idade <= 12 ? "criança" : "adolescente");
```

O uso de operadores ternários aninhados pode deixar o código ilegível, por isso, para manter uma boa reputação, não é recomendado que se faça isso.

7.26. Operadores de incremento e decremento

Os operadores de incremento e decremento também são conhecidos como acréscimo e decréscimo. Esses operadores aumentam ou diminuem o valor de uma variável em exatamente uma unidade. Eles são compostos por dois sinais `++` (adição) ou `--` (subtração).

O exemplo abaixo incrementa a variável `idade` em uma unidade, mudando o valor de 10 para 11.

```
int idade = 10;
idade++;
```

A instrução `idade++` é uma versão mais simples dos exemplos abaixo:

```
// é o mesmo que
idade = idade + 1;

// e também
idade += 1;
```

É importante entender que a forma pós-fixada (com `++` depois do nome da variável) incrementa o valor após usar a variável.

No exemplo abaixo, a variável `novaIdade` será atribuída com o valor 10 e logo em seguida o valor da variável `idade` será incrementado em uma unidade, mudando seu valor para 11.

```
int idade = 10;

// novaIdade recebe 10 e idade é incrementada para 11
int novaIdade = idade++;
```

Existe também a forma pré-fixada (com `++` antes do nome da variável), que incrementa o valor antes de usar a variável.

No próximo exemplo, a variável `idade` será incrementada e terá o valor igual a 11 antes de atribuir um valor para `novaIdade`, que também ficará com valor 11.

```
int idade = 10;

// idade é incrementada para 11 e novaIdade recebe 11
int novaIdade = ++idade;
```

Assim como existe o operador de incremento, você pode usar o operador de decremento para diminuir o valor da variável em uma unidade, podendo ser da forma pós-fixada ou pré-fixada.

```
int idade = 10;

// novaIdade recebe 10 e idade é decrementada para 9
int novaIdade = idade--;
```

```
// idade é decrementada para 8 e outra recebe 8
int outraIdade = --idade;
```

Os operadores de incremento e decremento são usados principalmente para controlar *loops* (laços), onde é necessário um contador para saber quantas vezes o bloco do laço foi executado.

7.27. Estrutura de controle **while**

O **while** é uma das formas de fazer *loops* (laços) em Java. *Loops* são usados para executar um bloco de código diversas vezes, dependendo de uma condição ser verdadeira.

O exemplo abaixo imprime todos os números em um intervalo informado pelo usuário.

```
Scanner entrada = new Scanner(System.in);

System.out.print("Digite o numero inicial: ");
int numeroInicial = entrada.nextInt();

System.out.print("Digite o numero final: ");
int numeroFinal = entrada.nextInt();

int numeroAtual = numeroInicial;

while (numeroAtual <= numeroFinal) {
    System.out.println(numeroAtual);
    numeroAtual++;
}
```

O bloco de código dentro do **while** será executado enquanto o valor da variável `numeroAtual` for menor ou igual ao valor da variável `numeroFinal`. Veja que usamos o operador de incremento para adicionar à variável `numeroAtual` uma unidade a cada execução do bloco de código do *loop*.

7.28. Estrutura de controle **do/while**

O *loop* do tipo **do/while** é parecido com o **while**, com a diferença que a condição do laço é testada somente após a execução do bloco de código. Por isso, o bloco de código do *loop* é executado pelo menos uma vez.

O exemplo abaixo solicita ao usuário a entrada de um número, sendo que se for digitado 0, o laço é encerrado (de acordo com a condição), mas se outro número for digitado, uma somatória é feita e o programa imprime o valor acumulado.

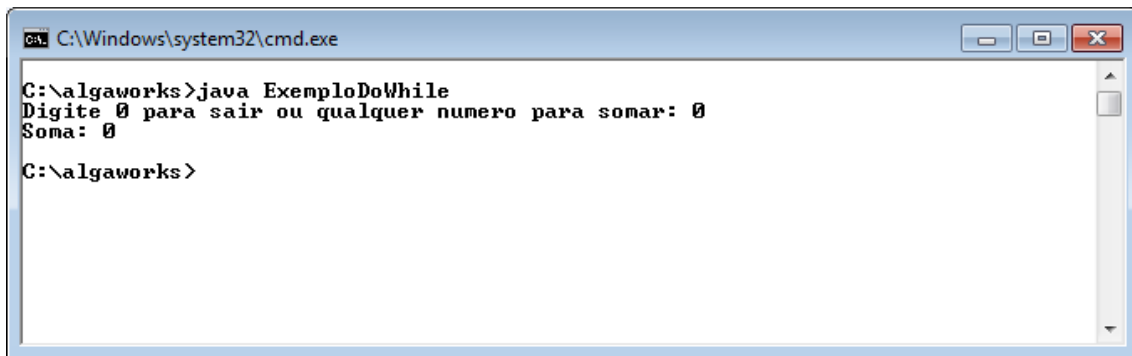
```
Scanner entrada = new Scanner(System.in);

int valor = 0;
int soma = 0;

do {
    System.out.print("Digite 0 para sair ou qualquer "
        + "numero para somar: ");
    valor = entrada.nextInt();
}
```

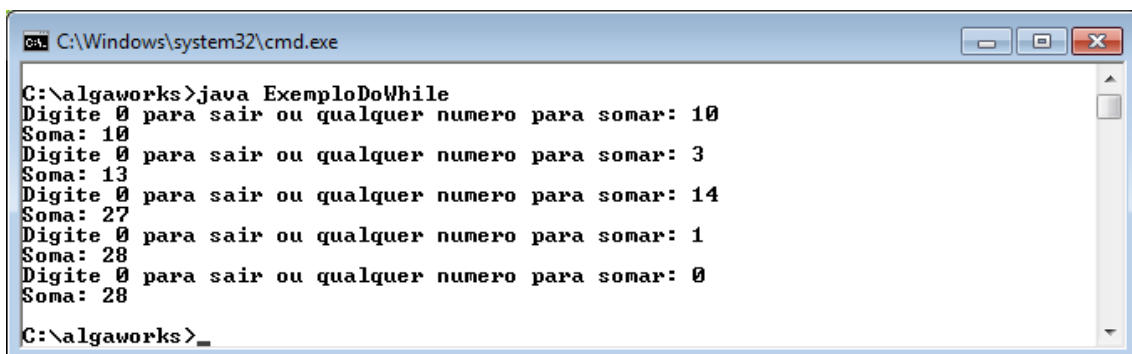
```
soma += valor;  
System.out.println("Soma: " + soma);  
} while (valor != 0);
```

Ao compilar e imprimir o programa, note que o bloco de código do `do/while` é executado pelo menos uma vez. Na execução abaixo, informamos o número 0 e saímos do laço imediatamente.



```
C:\Windows\system32\cmd.exe  
  
C:\algaworks>java ExemploDoWhile  
Digite 0 para sair ou qualquer numero para somar: 0  
Soma: 0  
C:\algaworks>
```

Na execução a seguir, informamos vários números antes de digitar o valor 0, por isso o laço foi executado diversas vezes.



```
C:\Windows\system32\cmd.exe  
  
C:\algaworks>java ExemploDoWhile  
Digite 0 para sair ou qualquer numero para somar: 10  
Soma: 10  
Digite 0 para sair ou qualquer numero para somar: 3  
Soma: 13  
Digite 0 para sair ou qualquer numero para somar: 14  
Soma: 27  
Digite 0 para sair ou qualquer numero para somar: 1  
Soma: 28  
Digite 0 para sair ou qualquer numero para somar: 0  
Soma: 28  
C:\algaworks>_
```

7.29. Estrutura de controle `for`

Uma das estruturas de controle mais usadas para fazer laços é o `for`, pois ela fornece uma forma fácil de iterar (percorrer) em um intervalo de valores. A forma básica da instrução `for` é a seguinte:

```
for (inicição; condição; incremento) {  
    // bloco de código do loop  
}
```

A expressão de *inicição* é executada uma única vez assim que o *loop* é iniciado. Normalmente é declarada uma variável de controle com um valor inicial.

A expressão de *condição* é semelhante às outras estruturas de controle. Você deve incluir uma expressão booleana que definirá a permanência ou término do laço. Essa expressão é analisada a cada iteração do *loop*.

A última expressão, de *incremento*, é executada após cada iteração do laço, e normalmente é usada para modificar o valor da variável de controle, seja incrementando ou decrementando.

Vamos a um exemplo:

```
Scanner entrada = new Scanner(System.in);

System.out.print("Ultimo numero: ");
int numeroFinal = entrada.nextInt();

for (int i = 1; i <= numeroFinal; i++)
    System.out.println(i);
}
```

O código acima executa o laço `for` e imprime números de 1 até um valor informado pelo usuário. Note que uma variável de controle `i` foi declarada na seção de *iniciação*, avaliada na *condição* e incrementada na seção de *incremento*.

7.30. Cláusulas `break` e `continue`

Quando trabalhamos com *loops*, pode surgir a necessidade de abandonar a execução do laço antes mesmo que a condição booleana seja falsa. Neste caso, devemos usar a cláusula `break`.

O exemplo abaixo solicita ao usuário um número (divisor) e imprime na tela de 100 até um número anterior ao primeiro múltiplo do valor informado (desde que seja até o número 200).

```
Scanner entrada = new Scanner(System.in);

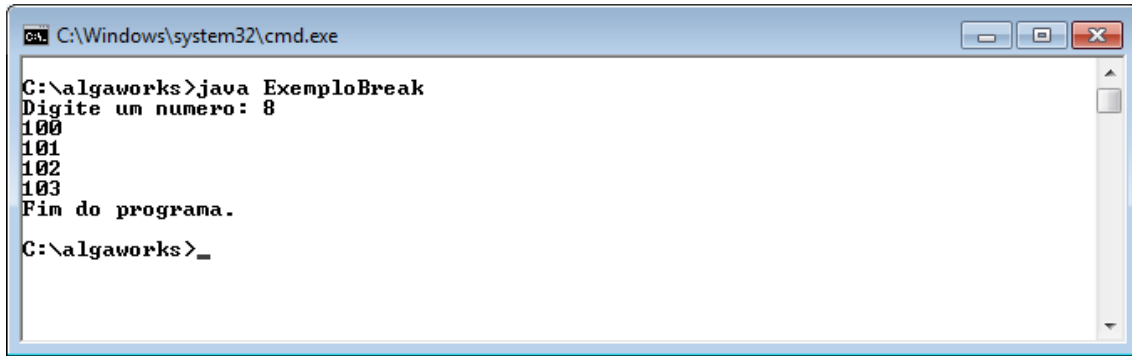
System.out.print("Digite um numero: ");
int divisor = entrada.nextInt();

for (int i = 100; i <= 200; i++) {
    if (i % divisor == 0) {
        break;
    }

    System.out.println(i);
}

System.out.println("Fim do programa.");
```

Na execução abaixo, informamos o número 8 e o programa exibiu os valores de 100 a 103, pois o número 104 é o primeiro múltiplo de 8 dentro do intervalo de 100 a 200. A cláusula `break` interrompeu o *loop*, mas não o programa.



```
C:\Windows\system32\cmd.exe

C:\algaworks>java ExemploBreak
Digite um numero: 8
100
101
102
103
Fim do programa.
C:\algaworks>
```

Outra necessidade que pode surgir durante o uso de laços é o abandono de apenas a iteração atual, mas não do *loop*. A cláusula responsável por fazer isso é a *continue*.

O exemplo abaixo é parecido com o anterior, porém ele imprime na tela todos os números entre 100 e 120, com exceção dos múltiplos do número informado.

```
Scanner entrada = new Scanner(System.in);

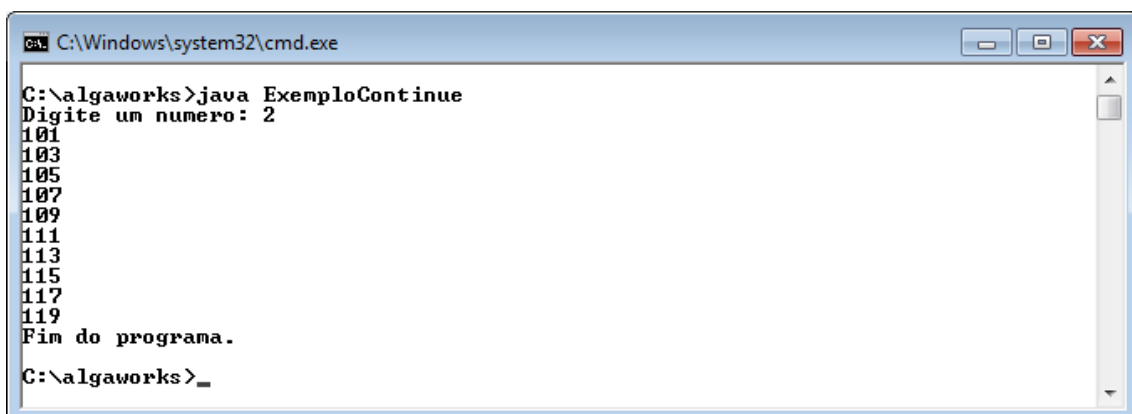
System.out.print("Digite um numero: ");
int divisor = entrada.nextInt();

for (int i = 100; i <= 120; i++) {
    if (i % divisor == 0) {
        continue;
    }

    System.out.println(i);
}

System.out.println("Fim do programa.");
```

Executamos o programa e informamos o número 2. Veja que foram exibidos apenas os números ímpares (não múltiplos de 2).



```
C:\Windows\system32\cmd.exe

C:\algaworks>java ExemploContinue
Digite um numero: 2
101
103
105
107
109
111
113
115
117
119
Fim do programa.
C:\algaworks>
```

Quando você está trabalhando com um laço dentro de outro, a chamada das cláusulas *break* e *continue* abandonam o laço ou a iteração do laço mais interno, a não ser que um rótulo seja especificado. Se precisar disso, pesquise na internet sobre cláusulas *break* e

`continue` rotuladas. O melhor seria você não usar essas cláusulas rotuladas e pensar em alguma lógica mais simples, pois elas podem deixar seu código-fonte difícil de ser compreendido.

8. Próximos capítulos

Esta apostila está incompleta (em desenvolvimento). Acesse o site <http://www.algaworks.com/ead> para acompanhar as novas versões.

9. Sites Interessantes

Blog da AlgaWorks

<http://www.algaworks.com/blog>

Java.NET

<http://www.java.net>

Grupo de Usuários Java

<http://www.guj.com.br>

JEE Brasil

<http://www.jeebrasil.com.br>

JavaFree – Se é Java, é Free!

<http://www.javafree.com.br>

Portal Java

<http://www.portaljava.com.br>

MundoOO

<http://www.mundooo.com.br>

[illegible]