

Curso Linguagem de programação



Java

Sumário

Introdução.....	1
Capítulo 1.O que é a linguagem java.....	2
1.1.- Histórico da linguagem.....	2
1.2.Uso da linguagem no mercado atualmente.....	3
1.3.Principais características da linguagem de programação java.....	3
1.4. Ambientes e ferramentas de desenvolvimento com java.....	7
1.4.1. Instalação do JDK ou SDK e JRE e suas configurações.....	7
1.5.Meu primeiro programa em java.....	7
1.5.1.Comentários em Java.....	9
1.5.2.Compilando meu primeiro programa em Java.....	10
1.6.Erros comum em programas java.....	12
1.6.1.Erros em java.....	12
1.6.2.Erro em tempo de compilação.....	12
1.6.3.Erro em tempo de execução.....	12
1.7. IDEs.....	13
1.7.1.Eclipse.....	13
1.7.2.NetBeans.....	14
1.7.3.IntelliJ.....	14
1.7.4.BlueJ.....	14
1.7.5.Jdeveloper.....	15
1.8.Palavras-chaves e identificadores.....	15
1.9.Tipos primitivos.....	16
1.9.1.Valores literais.....	17
1.9.1.1. Literais Inteiros.....	17
1.9.1.2. Literais fracionários, pontos flutuantes.....	17
1.9.1.3. Literais booleanos.....	18
1.9.1.4. Literais de caracteres.....	19
1.9.1.4. Literais para String.....	19
1.10. Operadores em Java.....	20
1.10.1.Operadores de atribuição.....	20
1.10.2. Operadores de aritméticos.....	21
1.10.3. Operadores unários.....	22
1.10.4. Operadores de atribuição composto.....	22
1.10.5. Operadores relacionais.....	23
1.10.6. Operadores lógicos.....	24
1.10.7. Operador ternário.....	25
1.11. Instanceof.....	26
1.12. Operador de concatenação de Strings.....	27
1.13. Precedência dos operadores.....	27
1.14. Exercícios propostos.....	28
Capítulo 2. Controle de fluxo.....	31
2.1.Estruturas de decisão.....	31
2.1.1. Estrutura de decisão if.....	31
2.1.1. Estrutura de decisão if-else.....	32
2.1.2. Estrutura de decisão if-else-if.....	32
2.1.3. Estrutura de decisão if-else aninhado.....	35
2.2. Estrutura de decisão SWITCH.....	36

2.2. Estruturas de repetição.....	37
2.2.1. Recursividade.....	38
2.2.2. Estrutura de repetição While.....	39
2.2.3. Estrutura de repetição Do While.....	41
2.2.4. Estrutura de repetição For.....	42
2.3. Exercícios propostos.....	44
Capítulo 3. Programação Orientada a objetos POO.....	46
3.1. Componentes da programação orientada a objetos.....	46
3.1.1. Classes.....	48
3.1.1.1. Atributos e métodos.....	49
3.1.1.2. Modificadores de acesso.....	49
3.1.1.3. Métodos construtores.....	50
3.1.1.4. Métodos acessores.....	50
3.1.1.5. Métodos.....	51
3.1.1.5.1. Método sem parâmetro e sem retorno.....	51
3.1.1.5.2. Método com parâmetro e com retorno.....	52
3.1.1.5.3. Método com parâmetro e sem retorno.....	52
3.1.2. Objetos.....	52
3.2. Herança	53
3.2.1. Classes e métodos abstratos.....	54
3.3. Interfaces e herança múltiplas.....	56
3.4. Exercícios propostos.....	57
Capítulo 4. Arrays, listas e coleções.....	59
4.1. Arrays.....	59
4.2. Array & ArrayList.....	60
4.3. Exercícios propostos.....	61
Capítulo 5. Git & GitHub.....	62
5.1. Principais comandos Git.....	62
5.2. Exercícios propostos.....	64
Capítulo 6. Sistema gerenciador de banco de dados SGBD MySQL.....	66
6.1. MySQL.....	66
6.2. Comandos do MySQL.....	66
6.3. Exercícios propostos.....	68

Introdução

Essa apostila tem como objetivo ajudar ao aluno na disciplina de Introdução a Programação Orientada a Objetos do nosso curso, visto que o requisito básico para que se possa alcançar todos os objetivos do nosso curso nesta disciplina é que o aluno tenha passado pela disciplina de Introdução a Lógica de Programação para Computadores. Mas não nos atearemos em barrá-los, sem o pré-requisito básico, isso porque confiamos e acreditamos na capacidade dos nossos alunos que com muitas força, dedicação e vontade de vencer na vida, conseguirá ultrapassar qualquer preconceito contra o acesso ao conhecimento tecnológico.

A linguagem Java é um Software Livre que é uma tendência e uma alternativa para o mundo corporativo no que se refere a softwares, e que também tem sido atualmente desenvolvido por programadores espalhados ao redor mundo, sendo que boa parte dessas tecnologias veem sendo desenvolvidas utilizando ferramentas livres na maioria de seus projetos, contribuindo para os avanços e colaborando para o acesso de milhares de pessoas as inclusões digitais no que tange as tecnologias da informação no mundo. Daremos também a nossa disciplina uma ênfase maior no uso desse tipo de tecnologia, passando para os nossos alunos um modelo e uma alternativa fantástica, que é a utilização das ferramentas livres disponíveis na internet.

Primeiramente vamos estudar um pouco sobre a história da linguagem de programação java que utilizaremos na nossa disciplina. Relatamos as características principais da linguagem que para muitos ainda é desconhecido, ou seja, conhecer a ferramenta antes de dominá-la é de extrema importância para os futuros programadores.

Conheceremos novas funcionalidades que foram implementadas a partir da versão 8 do Java assim com o Spring Boot e padrões de projeto. Tudo que é necessário para você desenvolver seus aplicativos tanto para Desktop quanto para a Web.

Capítulo 1. O que é linguagem Java.

Java não é somente uma linguagem de programação, mas sim uma tecnologia que em como objetivo construir programas que possam ser utilizados em qualquer plataforma, computador, ou dispositivo móvel, com o mínimo possível de dependência com um sistema operacional ou ambiente de desenvolvimento. Java possui uma sintaxe extremamente similar à do C++, e com diversas características herdadas de outras linguagens, como *Smalltalk* e *Modula3*. É antes de tudo é uma boa linguagem, simples, com um ambiente de execução de alta qualidade e uma vasta biblioteca de classes disponíveis. Essa combinação é que torna Java uma proposta irresistível para tantos programadores. Além de contar com as seguintes características, totalmente orientada a objetos, fortemente tipada, independente de arquitetura, robusta, segura, portátil, bem estruturada, suporta programação distribuída, *multithreaded* e conta com *garbage collection*. Atualmente Java é muito utilizado nos sites de grandes empresas e organizações governamentais, mesmo após a ascensão de outras linguagens como Python e Javascript o Java continua sendo amplamente utilizado com muita procura no mercado de trabalho .

1.1. - Histórico da linguagem

Em 1991, na empresa *Sun Microsystems* ,foi iniciado o *Green Project*, o berço do Java uma linguagem de programação orientada a objetos. Os mentores do projeto eram Patrick Naughton, Mike Sheridan, e James Gosling. O objetivo do projeto não era a criação de uma nova linguagem de programação, mas antecipar e planejar a “próxima onda” do mundo digital. Eles acreditavam que em algum tempo haveria uma convergência dos computadores com os equipamentos e eletrodomésticos comumente usados pelas pessoas no seu dia a dia.

.James Gosling especificou uma nova linguagem de programação para o seu novo projeto e decidiu batizá-la de “Oak”, que quer dizer carvalho, uma árvore que ele podia observar quando olhava pela sua janela. O próximo passo era encontrar um mercado para o *7. A equipe achava que uma boa ideia seria controlar televisões e vídeo por demanda com o equipamento. Eles construíram um demo chamado MovieWood, mas infelizmente era muito cedo para que o vídeo por demanda bem como as empresas de TV a cabo pudessem viabilizar o negócio. A ideia que o *7 tentava vender, hoje já é realidade em programas interativos e também na televisão digital. Permitir ao telespectador interagir com a emissora e com a programação em uma grande rede a cabos, era algo muito visionário e estava muito longe do que as empresas de TV a cabo tinham capacidade de entender e comprar.

O protótipo se chamava *7 (leia-se “StarSeven”), um controle remoto com uma interface gráfica touchscreen. Para o *7 foi criado um mascote, hoje amplamente conhecido no mundo Java, o Duke. O trabalho do Duke no *7 era ser um guia virtual ajudando e ensinando o usuário a utilizar o equipamento. O *7 tinha a habilidade de controlar diversos dispositivos e aplicações.

A ideia certa, na época errada. A sorte é que o bom da Internet aconteceu, e rapidamente uma grande rede interativa estava se estabelecendo. Era este tipo de rede interativa que a equipe do *7 estava tentando vender para as empresas de TV a cabo. E, da noite para o dia, não era mais necessário construir a infraestrutura para a rede, em um golpe de sorte, ela simplesmente estava lá. Gosling foi incumbido de adaptar o Oak para a Internet e em janeiro 1995 foi lançada uma nova versão do Oak que foi rebatizada para Java. A tecnologia Java tinha sido projetada para se mover por meio das redes de dispositivos heterogêneos, redes como a Internet. Agora aplicações poderiam ser executadas dentro dos Browsers nos Applets Java e tudo seria disponibilizado pela Internet instantaneamente. Foi o estático HTML dos Browsers que promoveu a rápida disseminação da dinâmica tecnologia Java. A velocidade dos acontecimentos seguintes foi assustadora, o número de usuários cresceu rapidamente, grandes players, como a IBM anunciaram suporte para a tecnologia Java.

1.2. Uso da linguagem no mercado atualmente.

Desde seu lançamento, em maio de 1995, a plataforma Java foi adotada mais rapidamente do que qualquer outra linguagem de programação na história da computação. Em 2003 Java atingiu a marca de 4 milhões de desenvolvedores em todo mundo. Java continuou e continua crescendo e hoje é com certeza um padrão para o mercado oferecendo qualidade, performance e segurança ainda sem nenhum competidor a altura. Atualmente Java é a escolha para construir sistemas robustos, confiáveis e distribuídos, rodando desde grandes servidores, por isso se tornou popular pelo seu uso na Internet e hoje possui seu ambiente de execução presente em web browsers, mainframes, SOs, celulares, e cartões inteligentes, entre outros dispositivos.

Atualmente o Java também é utilizado para aplicativos IoT (Internet das coisas) por ser uma linguagem multiplataforma, ser portátil e ter um vasto repositório de APIs. Temos também várias bibliotecas Java para machine learn (aprendizado de máquina) que é um seguimento de inteligência artificial, o DL4J.

Resumindo, podemos com Java criar aplicativos para Desktop, dispositivos móveis, redes domésticas, páginas web, inteligência artificial. Um mercado bem amplo e ainda em expansão. A nova aposta da comunidade Jakarta, promete levar a linguagem de programação a outros níveis de desempenho.

1.3. Principais características da linguagem Java.

ORIENTAÇÃO A OBJETOS: Java é uma linguagem orientada a objetos que segue a linha purista iniciada por *Smalltalk* que é considerada puramente O.O e que tudo nesta são objetos. Com a exceção dos tipos primitivos da linguagem (char, int, float, etc.), a maior parte dos elementos de um programa java são objetos.

No que rege esse paradigma, o projeto orientado a objetos é uma técnica de programação que se concentra nos dados(= objetos) e nas interfaces para este objeto. O código é organizado em classes, que podem estabelecer relacionamentos de herança simples entre si, somente a herança simples é permitida em java. Há uma forma de “simular” herança múltipla em Java com o uso de interfaces.

SIMPLICIDADE: Java, é muito parecida com C++,mas muito mais simples. Java não possui sobrecarga de operadores, structs, unions, aritmética de ponteiros, herança múltipla, arquivos.h, diretivas de préprocessamento e a memória alocada dinamicamente é gerenciada pela própria linguagem, que usa algoritmos de garbage collection para desalocar regiões de memória que não estão mais em uso. Outro aspecto da simplicidade do Java é o tamanho pequeno. Um dos objetivos do Java é permitir a construção de software que possa rodar independentemente em qualquer máquina de pequeno porte. O tamanho do interpretador básico e do suporte a classes é de cerca de 40K bytes; adicionando as bibliotecas básicas padrão e o suporte a linhas de execução (essencialmente um microkernel autocontido), têm-se outros 175 K.

SEGURA: Java foi elaborada para ser usada em ambientes de rede distribuída, por isso permite a construção de sistemas livres de vírus e intrusões. A presença de coleta automática de lixo, evita erros comuns que os programadores cometem quando são obrigados a gerenciar diretamente a memória (C, C++, Pascal).

A eliminação do uso de ponteiros, em favor do uso de vetores, objetos e outras estruturas substitutivas traz benefícios em termos de segurança. O programador é proibido de obter acesso à memória que não pertence ao seu programa, além de não ter chances de cometer erros comuns tais como "reference aliasing" e uso indevido de aritmética de ponteiros. Estas medidas são particularmente úteis quando pensarmos em aplicações comerciais desenvolvidas para a internet.

ROBUSTA: Java foi elaborada para a produção de programas que devam ser confiáveis em vários sentido. Java põe bastante ênfase na verificação rápida de possíveis problemas, na verificação dinâmica posterior(em tempo de execução),e em eliminar situações propensas a erros. A principal diferença entre Java e C/C++ é que Java possui um modelo de ponteiros que elimina a possibilidade de sobrescrever a memória e corromper dados.

PROCESSAMENTO DISTRIBUÍDO: A linguagem de programação Java possui uma extensa biblioteca de rotinas para lidar com protocolos TCP/Ip, como HTTP e FTP. As aplicações Java podem abrir e acessar objetos através da rede via Urls, com a mesma facilidade com que acessa um sistema de arquivos locais.

MULTITHREADING: Múltiplas linhas de execução. Em Java é uma surpresa agradável e fácil. Linhas de execução em Java podem também beneficiar-se de sistemas multiprocessadores se o sistema operacional de base o fizer. O lado ruim é que as implementações de linhas de execução nas plataformas mais importantes diferem radicalmente, e o Java não faz nenhum esforço para ser neutro em relação as plataformas nesse aspecto. Somente o código para chamar o *multithreading* permanece o mesmo para todas as máquinas.

EXCEÇÕES: Todo programador em geral está bastante acostumado com o computador "travando" por causa de um erro em um programa. Em C++, por exemplo, a simples tentativa de abertura de um arquivo inexistente pode obrigar ao programador a reiniciar o computador. Programas Java, contudo, não "dão pau" no computador, já que a

máquina virtual Java faz uma verificação em tempo de execução quanto aos acessos de memória, abertura de arquivos e uma série de eventos que podem gerar uma "travada" em outras linguagens, mas que geram exceções em programas Java. Em geral, ao escrever programas Java utilizando-se de herança de classes predefinidas, força-se em geral ao programador escrever algumas rotinas de tratamento de exceção, um trabalho que, se de início pode parecer forçado, poupará o programador de bastante dor de cabeça no futuro.

GARBAGE COLLECTOR: Em Java, os programadores não necessitam se preocupar com o gerenciamento de memória como em C++. Em C++, todo bloco de memória alocado dinamicamente (com **new**, **malloc** ou função similar) deveria ser liberado quando não fosse mais usado (com **free**, **delete** e parentes próximos). Isso acarretava diversos problemas mesmo ao programador mais experiente, que tinha que manter sempre um controle das áreas de memória alocadas para poder liberá-las em seguida. Java, ao contrário, utiliza-se de um conceito já explorado pela linguagem Smalltalk, que é o de garbage collection (coleta de lixo). Sua função é a de varrer a memória de tempos em tempos, liberando automaticamente os blocos que não estão sendo utilizados. Se por um lado isso pode deixar o aplicativo um pouco mais lento, por manter uma thread paralela que dura todo o tempo de execução do programa, evita problemas como referências perdidas e avisos de falta de memória quando se sabe que há megas e megas disponíveis na máquina.

COMPILADA E INTERPRETADA: Uma das características de Java que a tornou ideal para seu uso na elaboração de aplicativos distribuídos foi a sua independência de plataforma. Isso porque o compilador gera um formato de arquivo de objetos neutro em relação a arquitetura, o código compilado é executável em vários processadores, desde que haja a presença do sistema em tempo de execução Java. O compilador java consegue isso gerando instruções bytecode que não tem nada a ver com uma arquitetura computacional específica. Ao contrário eles são elaborados para ser de fácil interpretação em qualquer máquina e de fácil tradução para código de máquina nativo imediatamente.

MÁQUINA VIRTUAL: É uma máquina imaginária que é implementada através de um software emulador em uma máquina real. A JVM provê especificações de plataforma de hardware na qual compila-se todo código de tecnologia Java. Essas especificações permitem que o software Java seja uma plataforma independente pois a compilação é feita por uma máquina genérica conhecida como JVM.

PORTÁVEL: Ao contrário do C e C++ que não existem aspectos da especificação que sejam dependentes da implementação. Os tamanhos dos tipos de dados primitivos são especificados, bem como o comportamento da aritmética neles. Por exemplo em Java um **int** é sempre um número inteiro de 32 bits, enquanto que em C/C++, **int** pode significar um inteiro de 16 bits ou um inteiro de 32 bits. Com o tamanho fixo para tipos numéricos elimina-se a causa de grande dores de cabeça relacionadas a portabilidade. A portabilidade em java é atingida através da utilização de bytecodes. Bytecode é um formato de código intermediário entre o código fonte, o texto que o programador consegue manipular, e o código de máquina, que o computador consegue executar. Na plataforma Java, o bytecode é interpretado por uma máquina virtual Java. A portabilidade do código Java é obtida à medida que máquinas virtuais Java estão disponíveis para diferentes

plataformas. Assim, o código java que foi compilado em uma máquina pode ser executado em qualquer máquina virtual Java, independentemente de qual seja o sistema operacional ou o processador que executa o código.

JAVA E A INTERNET: Os programas java que rodam em páginas web são chamados applets. Para usar applets, você precisa de um navegador web habilitado para Java, o qual executará os bytecodes para você. Em particular servidores de aplicação podem usar as capacidades de monitoramento da máquina virtual java para realizar o equilíbrio automático de carga, controle de conexões a banco de dados, sincronização de objetos, desligamento e reinicialização seguros, além de outros serviços necessários para aplicações escaláveis de serviços, mas que são notoriamente difíceis de se implementar corretamente. Em java, é possível escrever aplicações completas, inclusive acessando bancos de dados relacionais independentemente do servidor web, bem como é possível implementar os níveis de interface com o usuário e de lógica do negócio, utilizando um servidor de banco de dados para implementar o nível de acesso aos dados.

Distribuição da tecnologia java: O **Java SE** (Java 2 Standard Edition) ou Java SE é uma ferramenta de desenvolvimento para a plataforma Java. Ela contém todo o ambiente necessário para a criação e execução de aplicações Java, incluindo a máquina virtual Java (JVM), o compilador Java, as APIs do Java e outras ferramentas utilitárias.

Java ME, java Plataforma, Micro Edition (Java ME), ainda conhecida por J2ME, é uma tecnologia que possibilita o desenvolvimento de software para sistemas e aplicações embarcadas, ou seja, toda aquela que roda em um dispositivo de propósito específico, desempenhando alguma tarefa que seja útil para o dispositivo. É a plataforma Java para dispositivos compactos, smartwatches, smart tvs, celulares e uma outra gama de dispositivos.

Java EE (ou J2EE, ou Java 2 Enterprise Edition, ou em português Java Edição Empresarial) é uma plataforma de programação de computadores que faz parte da plataforma Java. O JEE (**J**ava **E**nterprise **E**dition) é a plataforma Java voltada para redes, internet, intranets e afins. Assim, ela contém bibliotecas especialmente desenvolvidas para o acesso a servidores, a sistemas de email, a banco de dados, etc. Por essas características, o JEE foi desenvolvido para suportar uma grande quantidade de usuários simultâneos. A plataforma JEE contém uma série de especificações, cada uma com funcionalidades distintas. Que são:

- **JDBC (Java Database Connectivity)**, utilizado no acesso a banco de dados.
- **JSP (Java Server Pages)**, um tipo de servidor Web. Os servidores web são as aplicações que permitem a você acessar um site na internet.
- **Servlets**, para o desenvolvimento de aplicações Web, isto é, esse recurso "estende" o funcionamento dos servidores web, permitindo a geração de conteúdo dinâmico nos sites.

Atualmente o Java EE vem cedendo espaço para o Jakarta EE, uma vez que a Oracle cedeu o mesmo para a Eclipse Foundation, responsável pela IDE de mesmo

nome. O que nos leva a um novo nível de programação e uma atualização exponencial da linguagem feito pela comunidade. Com o código do Java sendo aberto nós tivemos um enorme avanço na linguagem.

1.4. Ambientes e ferramentas de desenvolvimento Java

A tecnologia java fornece como ambiente de desenvolvimento um grande conjunto de ferramentas que engloba: um compilador, um interpretador, um gerador de documentação, ferramenta de empacotamento de classes de arquivos e outros.

1.4.1. Instalação do JDK e JRE.

A Sun a empresa proprietária do Java que atualmente pertence a empresa (Oracle). Ela disponibiliza, basicamente, duas versões de sua máquina virtual: o JRE (Java Runtime Environment) e JDK (Java Development Kit). O JRE contém a máquina virtual que permite rodar programas feitos na linguagem Java em uma máquina. O JDK, por sua vez, é um pacote para desenvolvedores programarem suas aplicações em Java, possuindo vários utilitários, inclusive compilador e bibliotecas. Uma das versões mais usadas do Java foi a 1.4.x, que possuía inclusive um visual um pouco (para ser modesto) rústico. Mais para frente a Sun lançou a versão 1.5.0, com várias melhorias em termos de programação e um visual um pouco melhor. Foi lançado em dezembro de 2006 o Java 1.6.0, conhecido também como Java 6, que trouxe, finalmente, uma integração total com o desktop em termos visuais, e várias melhorias também para quem é programador.

A versão 8 do Java foi lançada em março de 2014 e trouxe muitas atualizações e deixou a linguagem bem mais fluída com as expressões Lambda e uma nova API de datas que deixou sua utilização mais natural e fácil de entender. Hoje em dia temos a versão Java Platform, Standard Edition 15 como a última a ser lançada.

1.5. Meu primeiro programa em Java

Como java é uma linguagem que é primeiramente compilada e em seguida interpretada. Para compilar e executar um programa escrito nessa linguagem devemos o primeiro construir o arquivo como o código fonte, que deve ter como extensão .java. Que no caso do exemplo o arquivo foi chamado de HelloWorld.java. Em seguida, o código fonte é compilado e um programa fonte em bytecodes (HelloWorld.class) é gerado. Durante a compilação, há uma checagem de erros do código fonte. O fonte em bytecodes só será gerado se nenhum erro tiver sido detectado. Por fim, qualquer dispositivo que execute java será capaz de interpretar este novo arquivo fonte e executar a aplicação. Os bytecodes são lidos e executados (ou seja, interpretados) pela Máquina Virtual Java (JVM – Java Virtual Machine) em um computador, em um celular, etc., além de serem independentes de plataforma.

A geração dos bytecodes ou, para o exemplo, do arquivo HelloWorld.class é feita a partir da execução do comando **javac HelloWorld.java**. O próximo passo é fazer com que

a JVM execute o arquivo HelloWorld.java, através do comando **java HelloWorld.class**. Dessa maneira, a JVM traduz o código compilado para uma linguagem que a máquina entenda e o programa é executado, por isto os programas em java podem executar em qualquer plataforma de hardware ou software que possua uma versão da JVM. A Máquina Virtual java é definida como uma máquina imaginária implementada através da emulação em um software executado em uma máquina real. Ela possui uma arquitetura que permite garantir segurança. Quando um programa Java é executado, seus bytecodes são verificados pela JVM para que estejam de acordo com os seus requisitos de segurança, impedindo a execução de código com alguma irregularidade. Assim, códigos fonte com instruções que acessem áreas restritas da memória ou até mesmo recursos do hardware não são executados pela JVM.

```
1 public class OlaMundo {
2     //Programa inicial que imprime na tela "Olá Mundo!"
3     public static void main(String Args[]){
4         System.out.println("Olá mundo!");
5     }
6
7 }
```

A palavra **public** indica que a classe terá um nível de acesso público, ou seja, que ela será acessível por qualquer classe, dentro e fora do projeto. A palavra **class** indica que uma classe está sendo declarada e seu nome é **HelloWorld**. A **{** delimita o limite inicial da classe.

```
1 public class OlaMundo {
```

A palavra **public** já foi descrita, portanto permite que o método seja acessado publicamente. A palavra **static** será descrita posteriormente. O lugar onde fica a palavra **void** é onde se deve indicar o tipo de retorno do método. Neste caso, não retorna nenhum valor, dessa maneira o método é declarado como **void**. O conjunto **String[] args** presentes entre () são os argumentos do método. Neste exemplo, o método possui um único argumento (um vetor de Strings denominado args). A **{** delimita o limite inicial do método.

A terceira linha consiste no conteúdo do método principal, formado apenas por um comando, ou uma instrução que é composta de uma ou mais linhas terminadas por ponto e vírgula. Exemplo:

```
System.out.println("Olá mundo!");
```

O **System.out.println** é usado para exibir algo na saída padrão, por padrão, na linha de comando. Será exibido o que tiver entre (""), no caso, **Hello world**. No final deste

comando, tem se um; que o finaliza. Por fim, há duas “fecha chaves” `}}`, que delimitam o fim do método e da classe, respectivamente.

```
3 public static void main(String Args[]) {  
4     System.out.print("Olá ");  
5     System.out.println("mundo!");  
6 }
```

Na imagem acima nós temos o `public static void main` é o método principal que será chamado sem que a classe seja instanciada, graças ao `static`, com o retorno vazio, definido pelo `void`. Dentro do método temos duas linhas de código que serão responsáveis por imprimir “Olá mundo”. O `print` imprime a palavra e mantém o cursor na mesma linha, o `println` faz com que o mesmo salte para a próxima linha. Vocês entenderão cada denominação dessas mais para frente.

Blocos de código: Um bloco é formado por uma ou mais instruções agrupadas entre chaves indicando que formam uma só unidade. Blocos podem ser organizados em estruturas aninhadas Indefinidamente.

Qualquer quantidade de espaços em branco é permitida. Um exemplo de bloco código mostrado ao lado ilustra perfeitamente como são organizadas as estruturas de blocos.

1.5.1 Comentários em Java.

Comentários são notas escritas pelo programador para fins de documentação. Estas notas não fazem parte do programa e não afetam o fluxo de controle. Java suporta três tipos de comentários: comentário de linha estilo C++, comentário de bloco estilo C e um comentário estilo Javadoc (utilizado compor a documentação do programa).

Comentário de linha: Comentários com estilo em C++ se iniciam por `///`. Todo e qualquer texto colocado após as `//` é ignorado pelo compilador e tratado como comentário. Por exemplo:

```
2 //Comentário de uma única linha!
```

Comentário de bloco: Comentários com estilo em C, também chamados de comentários multilinhas, se iniciam com `/*` e terminam com `*/`. Todo o texto posto entre os dois delimitadores é tratado como comentário. Diferente do comentário estilo C++, este pode se expandir para várias linhas. Por exemplo:

```
4  /*
5
6      Este é um comentário de várias linhas.
7      Um tipo especial e mais detalhado de se fazer comentários.
8      É delimitado pelos caracteres / * * /
9
10 */
```

Comentário estilo Javadoc: Este comentário é utilizado na geração da documentação em HTML dos programas escritos em Java. Para se criar um comentário em estilo Javadoc deve se iniciar o comentário com `/**` e terminá-lo com `*/`. Assim como os comentários estilo C, este também pode conter várias linhas. Este comentário também pode conter certas tags que dão mais informações à documentação. Por exemplo:

```
13 /**
14  * Comentário de documentação do programa escrito em Java
15  * Autor: UTD
16  * Data: 01/01/2021
17  *
18  */
```

1.5.2. Compilando o primeiro programa.

Após instalar o JDK, que contém uma JRE e colocar o caminho da pasta no PATH das variáveis de ambiente podemos executar o comando JAVAC para criar os nossos arquivos .class para serem executados.

Primeiro criaremos um arquivo de texto, pode até ser no bloco de notas, com o nome da classe que será criada e a extensão .java. Por exemplo nosso primeiro programa será o **OiMundo.java** é muito importante que o nome do arquivo seja exatamente igual ao da classe contida nele. Outra observação importante é que OiMundo está no estilo CascalCase, onde cada palavra se inicia com letra maiúscula e são todas juntas. O nosso programa será criado da maneira que é ilustrado na imagem abaixo.

```
1  /**
2   * Programa que imprime olá mundo.
3   *
4   * versão: 1.0
5   *
6   * data: 26/10/19
7   *
8   * author: Instrutor.
9   */
10
11 public class OiMundo{
12
13     //método principal de execução da classe.
14     public static void main(String[] args){
15         System.out.println("Olá mundo!");
16     }
17
18 }
```

Agora no prompt de comando navegamos até a pasta que está o nosso arquivo e executamos o comando `javac OiMundo.java`. Ao executar este comando e supondo que não aja nenhum erro no arquivo `.java` o compilador vai criar um arquivo `.class` que caso você abra com um editor de texto, estará em uma linguagem irreconhecível, essas linhas de código serão entendidas e executadas pela nossa JRE que retornará o que foi especificado dentro do método `public static void main`. Para tal execute agora o comando `java OiMundo`, fica a sua escolha colocar o `.class` ou não o programa será executado da mesma forma, só não coloque a extensão errada, porque acarretará em erro.



```
C:\Windows\system32\cmd.exe
C:\UTD>javac OiMundo.java_
```



```
C:\Windows\system32\cmd.exe
C:\UTD>javac OiMundo.java
C:\UTD>java OiMundo
Olá! mundo!
C:\UTD>
```

Se você notar o Olá saiu meio esquisito, isso acontece porque o prompt de comando ao retornar o print não reconheceu o A com acento e colocou o seu correspondente na tabela unicode.

Cuidados importantes na hora de executar um programa Java.

- Sempre que abrir um bloco de código { é uma boa prática fechá-lo antes de inserir o código dentro para que posteriormente não aja problemas com os mesmos.
- Tenha muito cuidado e certifique-se de encerrar seus comandos com ; é um erro muito comum deixar de colocá-los.
- Certifique-se que o arquivo **.java** tenha o mesmo nome que a classe principal contida nele.
- O java é case sensitive o que significa que ele diferencia letras maiúsculas de minúsculas, ou seja, Camelo não é igual a camelo e tão pouco caMelo.
- Verifique a versão instalada do JDK, e se o java foi adicionado corretamente ao PATH das variáveis de ambiente.

- A utilização de uma IDE irá nos ajudar bastante com os erros e criação de código, como será visto mais a frente.
- Aprenda a ler o erro que aparece no console para, com calma, tentar entender o que houve como resolver.

1.6. Erros comuns em programa java

1.6.1. Erros em java:

Durante o desenvolvimento de programas, é muito comum que erros podem ocorrer, em java pode ser em dois domínios: tempo de compilação e tempo de execução.

1.6.2. Erro em tempo de compilação:

Erros ocorridos durante a fase de compilação ocorrem quando se executa o **javac**, e são fáceis de corrigir. Há dois tipos:

- Erros de processamento do arquivo (parsing): ponto e vírgula faltando, parênteses, aspas, chaves ou colchetes descasados. Identifica apenas o arquivo e a linha onde o erro pode ter iniciado.
- Um erro causa vários outros e nem sempre a mensagem é precisa. Erros de compilação do código, realizada depois do parsing: além da linha e do arquivo, identificam a classe e método. Geralmente as mensagens são bastante elucidativas.

1.6.3. Erro em tempo de execução:

Erros que ocorrem durante o tempo de execução (runtime) ocorrem quando se executa o interpretador **java**, e são muito mais difíceis de localizar e consertar.

- Exception in thread "main": NoClassDefFoundError: Classe: a classe "Classe" não foi encontrada no CLASSPATH.
- Exception in thread "main": NoSuchMethodError: main: o sistema tentou chamar main() mas não o encontrou.
- ArrayIndexOutOfBoundsException: programa tentou acessar vetor além dos limites definidos, ou negativo.
- NullPointerException: referência para objeto é nula, ou variável de um tipo objeto foi declarada mas não inicializada.

1.7. Principais IDEs para Java.

Essa sigla significa Ambiente de desenvolvimento integrado é um software desenvolvido para facilitar a vida do programador oferecendo um GUI (grafic user interface).

Para facilitar o desenvolvimento de códigos, você pode simplesmente baixar o Eclipse, por exemplo, e ter um programa que já cria as pastas, organiza e informa sobre os possíveis erros que haverão na hora de compilar o código e soluções que podem ser aplicadas para resolvê-los, tem recursos de criação de códigos padrão, etc.

As IDEs também costumam contar com vários plugins que vão facilitar bastante a vida do programador aumentando a sua produtividade. Nesse tópico falaremos dos mais populares como Eclipse, Netbeans e IntelliJ, mas ainda existem Jcreator, Xcode, DrJava, Greenfoot, Jgrasp entre outros.

1.7.1. Eclipse

Eclipse já é bem notório no mundo da programação e é amplamente utilizado pela comunidade Java tendo sua versão para Desktop e online na nuvem, o Eclipse Che. Essa IDE vem com várias funcionalidade e plugins que podem ser customizados pelo desenvolvedor e instalados via market place.

Essa IDE vem com múltiplas perspectivas dentre as quais você pode ficar alternando dependendo da aplicação que está sendo feita também contando com vários softwares que vão te ajudar no seu trabalho como TomCat, glassfish, windowbuilder e ferramenta para modelagem de diagramas UML, testes de erros e muito mais. É uma IDE gratuita e pode ser baixada no site do eclipse.org para as arquiteturas X86 e X64, vale ressaltar que as novas atualizações do java são feitas para os 64 bits.



O Eclipse também possui suporte para várias outras linguagens de programação como Python, C, C++, Perl, Ruby, PHP, Scala, Javascript, Julia, o que torna o Eclipse não só uma poderosa ferramenta de programação Java como também uma IDE extremamente versátil.

1.7.2. NetBeans

Uma outra IDE bem difundida no mundo Java é o NetBeans e considerada oficial para o Java 8. Oferece ferramentas poderosas para o desenvolvimento Desktop, web e mobile. Encontra-se disponível para várias plataformas, cada nova versão do NetBeans vem com grandes inovações dos editores Java.



Temos um grande foco na sintática e semântica da linguagem o que facilita na produção de aplicativos customizados com refatoração do código livre de bugs.

Assim como o Eclipse essa IDE é muito versátil e cujas extensões também funcionam para C, C++, HTML 5, PHP, entre outros.

1.7.3. IntelliJ IDEA



A terceira entre as grandes IDEs está disponível em dois editores a Apache 2 Licensed community edition e uma edição comercial proprietária. Para uma melhor imersão do programador na codificação Java, O IntelliJ oferece refatoração entre as linguagens e análise de fluxo de dados, facilitando a vida do desenvolvedor com vários outros recursos e ferramentas. Essa IDE também dá suporte a várias outras linguagens de programação baseadas na JVM como Kotlin e Scala.

1.7.4. BlueJ

BlueJ foi projetado e implementado pelas equipes da universidade Monash, uma universidade pública de Melbourne, Austrália, e desempenha muito bem o papel para o qual foi produzida, é uma Java IDE mais voltada para o aprendizado acadêmico do paradigma de orientação a objetos e também é utilizado para projetos de pequeno porte. Apesar de ser desenvolvido para os iniciantes aprenderam a



programar em Java, muitos programadores experientes preferem usá-la já que permite ter uma visão e controle sobre os objetos e classes assim como sua estrutura.

1.7.5. JDeveloper (Oracle)

É um IDE gratuito disponibilizado pela Oracle que oferece funcionalidade para desenvolvimento em Java, Javascript, PHP, HTML, XML. JDeveloper abrange todo o ciclo de programação de um projeto, desde o design, passando pela elaboração do código até o deploy do projeto



1.8. Palavras chave e identificadores

Java é uma linguagem centrada nos pacotes; os desenvolvedores da linguagem assumiram que, para uma boa organização de nomes, seria fundamental na estrutura de programas em java. Imagine o seguinte pesadelo: três programadores, na mesma empresa, mas trabalhando em diferentes partes de um projeto, escrevem cada um uma classe chamada Utilitários. Se essas três classes Utilitários não tiverem sido declaradas em nenhum pacote explícito, e estiverem no classpath, não seria possível dizer ao compilador ou à JVM qual das três classes está tentando referenciar.

Identificadores legais são regras que o compilador usa para determinar se um dado nome é legal. Em Java, os identificadores devem ser compostos apenas por caracteres. Unicode, números, símbolos de moedas e caracteres de conexão como underscore. Regras válidas:

- Devem iniciar com uma letra, símbolo comercial cifrão (\$) ou underscore (_);
- Após o primeiro carácter podem ter qualquer combinação de letras, caracteres e números;
- Não possuem limite de tamanho;
- Não podem ser palavras reservadas restritas da linguagem;
- Em Java os identificadores são **case-sensitive** isto é, “JAVA” é totalmente diferente de “java” isso porque o compilador diferencia maiúsculos de minúsculos.

Palavras-chave: São identificadores que, em Java foram pré-definidas com propósitos bem definidos. Sendo assim não se podem usar esses identificadores para qualquer outra funcionalidade que não seja a que ele foi criado para executar.

STRING	BOOLEAN	WHILE	TRY	IMPORT
EXTENDS	FINAL	CONTINUE	INT	FLOAT
ELSE	SWITCH	BYTE	SUPER	THIS
ABSTRACT	PUBLIC	PRIVATE	CASE	ENUM
PROTECTED	INTERFACE	INTERGER	LONG	DOUBLE

1.9. Tipos primitivos.

Todo programa de computador deve ser capaz de lidar com dados para conseguir fazer seus processos como, por exemplo, somar, multiplicar, dividir, etc. Usar atributos é a melhor forma de manipular os dados. Os tipos de dados são o que definem a quantidade de memória do computador que será utilizado para guardar tal dado. As variáveis primitivas pode ser declaradas como variáveis de classe(static), variáveis instâncias, parâmetros de métodos ou variáveis locais. Uma vez declarado, seu tipo primitivo não pode nunca ser modificado, embora na maioria dos casos o seu valor possa se modificar. Java é uma linguagem fortemente tipada. Isso significa que toda variável deve ter um tipo declarado. Existem oito tipos primitivos em java. Quatro deles são do tipo inteiros; dois são do tipo de números de ponto flutuante; um é o tipo de caracteres char, usado para unidades de código no esquema de codificação Unicode e um deles é do tipo lógico para valores true/false (verdadeiro/falso).

Tipos	Primitivo	Valores, menor e maior		Tamanho
Booleano	Boolean	false	true	1 bit
Inteiro	byte	-128	127	8 bits
	short	-32768	32767	16 bits
	int	-2.147.483.648	2.147.483.647	32 bits
	long	-9.223.372.036.854.770.000	9.223.372.036.854.770.000	64 bits
Ponto Flutuante	float	-1,4024E-37	3.40282347E+38	32 bits
	double	-4,94E-307	1.79769313486231570E+308	64 bits
Caractere	char	0	65535	16 bits

1.9.1. Valores literais

Literais são a representação dos dados na linguagem, ou seja, é a representação dos valores dos tipos primitivos. Como números, caracteres true ou false no caso dos booleanos.

1.9.1.1. Literais inteiros

Na linguagem Java os literais inteiros são representados na base decimal que é a base que nós aprendemos durante o fundamental I na escola e que comumente usamos, na base octal e hexadecimal. Essas são as formas de se representar números inteiros no Java.

A base decimal é a mais conhecida usa os números 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 para compor os valores como por exemplo: 237 ou 1041. Os mesmos não necessitam de nenhum prefixo para serem utilizados.

A base octal utiliza os números de 0 a 7. Pegando o exemplo em decimal 2958 para octal fica: 5616, observe que na base octal, você nunca verá os algarismos 8 e 9.

A base hexadecimal utiliza os caracteres 0,1,2,3,4,5,6,7,8,9, A, B, C, D, E, F, as letras podem ser maiúsculas ou minúsculas. Ainda com o exemplo de 2958, transformando-o para a base hexadecimal, teremos: b8e.

1.9.1.2. Literais fracionários, pontos flutuantes

São usados para representar os números fracionários. Note que no inglês substituímos o ponto pela vírgula e vice-versa. Então o número 5,7 (cinco inteiros e sete décimos) na linguagem java é representado por 5.7, outra maneira de representar os números flutuantes é usando a notação científica, por exemplo 2.41e-4 é igual a 0,000241.

Os números de ponto flutuante são utilizados quando se busca obter uma precisão de calculo maior para casos onde as casas decimais fazem a diferença na resposta por isso que esses tipos primitivos ocupam mais espaço na memória do dispositivo, sendo o tipo double o maior dentre todos, cai entre nós, você já precisou usar um número tão grande quanto esse **1.79769313486231570E+308**?

Temos dois tipos de literais o float e o double. Para representarmos o float o número precisa ser sucedido por um **f**, caso o contrário o compilador pensará que o literal é do tipo **double** e vai imprimir um erro de typse mismatch ao tentar converter para o float. Sendo assim ao declarar uma variável do tipo float deve ser feito como é mostrado na figura.

```
// o primeiro número apresentará um erro pela falta do sufixo.  
float numero = 1.9;  
  
// o segundo número está descrito da maneira correta.  
float numero2 = 5.7f;
```

Ao contrário do float double não precisa do sufixo, mas caso queira identificar podemos pôr o sufixo **d** após o número, também vale ressaltar que se você pôr o **f** após o número além de não dar erro o número guardado no espaço de memória terá mais casas decimais. Como mostrado no exemplo.

```
22 public static void main(String[] args) {  
23  
24  
25     double duplo = 1.9;  
26  
27     double duplo2 = 5.7f;  
28  
29     double duplo3 = 8.4d;  
30  
31  
32     System.out.println("1º Exemplo:"+duplo);  
33  
34     System.out.println("2º Exemplo:"+duplo2);  
35  
36     System.out.println("3º Exemplo:"+duplo3);  
37 }  
38
```

Problems Javadoc Declaration Console Progress
<terminated> OlaMundo [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe
1º Exemplo:1.9
2º Exemplo:5.699999809265137
3º Exemplo:8.4

1.9.1.3. Literais booleanos

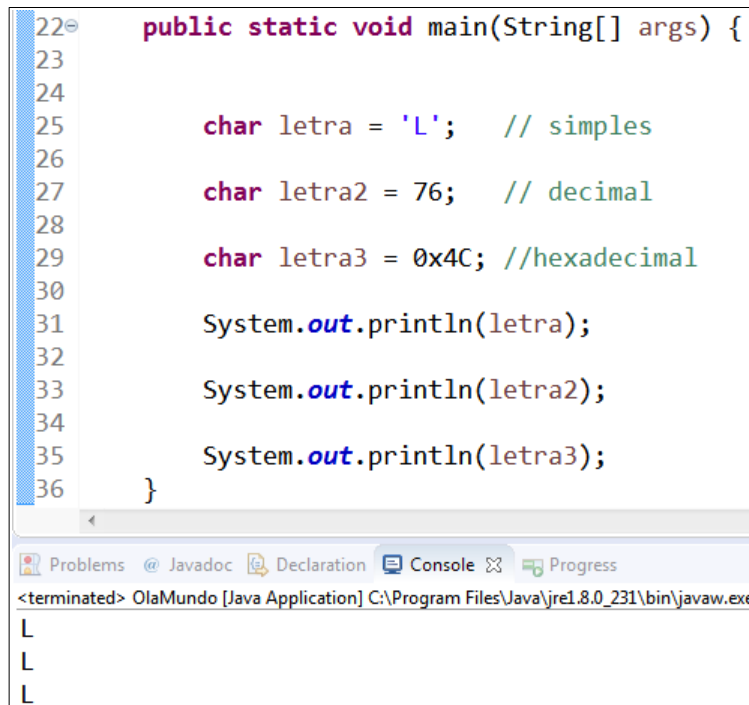
Os literais booleanos são auto-descritivos eles podem ser representados por dois valores **true** para verdadeiro e **false** para falso. São amplamente utilizados em estruturas de decisão e outras funcionalidades.

```
22 public static void main(String[] args) {  
23  
24  
25     boolean verdadeiro = true;  
26  
27     boolean falso = false;  
28  
29  
30     System.out.println(verdadeiro);  
31  
32     System.out.println(falso);  
33 }  
34  
35 }  
36
```

Problems Javadoc Declaration Console Progress
<terminated> OlaMundo [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe
true
false

1.9.1.4. Literais caracteres

O literal char é representado por um único caractere entre aspas simples. Eles podem também representar valores Unicode do caractere, usando a notação Unicode que acrescenta o prefixo \u ao valor, atribuir um literal numérico, contanto que ele esteja no intervalo de 16 bits sem sinal (65535 ou menor). Abaixo veremos três formas diferentes de imprimir a letra L.



```
22 public static void main(String[] args) {
23
24
25     char letra = 'L';    // simples
26
27     char letra2 = 76;    // decimal
28
29     char letra3 = 0x4C;  //hexadecimal
30
31     System.out.println(letra);
32
33     System.out.println(letra2);
34
35     System.out.println(letra3);
36 }
```

The screenshot shows an IDE window with the following tabs: Problems, Javadoc, Declaration, Console, and Progress. The Console output displays three lines of the letter 'L', corresponding to the three print statements in the code above.

1.9.1.5. Literais para String

O literal de String é um conjunto de caracteres que formam um objeto cujo objetivo é representar uma palavra. Apesar de não ser um tipo primitivo, incluímos String a este capítulo por serem usadas para representar valores literais. O que caracteriza uma String é o texto envolto de aspas duplas. Um número pode ser uma String, um texto vazio, um caractere e palavras simples até orações completas.

Na imagem abaixo temos alguns exemplos de quais valores uma variável do tipo String pode ter e fizemos até alguns testes, um deles foi mostrar como para o Java, letras maiúsculas e minúsculas se diferenciam fazendo com que o tipo String não seja igual a variável string. Obs: não é uma boa prática de programação dar um nome tão genérico para as suas variáveis, estamos fazendo isso para que a didática seja mais simples.

```
25 String string = null; // valor nulo
26
27 String string2 = ""; // string vazia
28
29 String string3 = "a"; // string composta de um caractere apenas
30
31 String string4 = "O rato roeu a roupa do rei de Roma."; // oração.
32
33 System.out.println(string);
34
35 System.out.println(string2);
36
37 System.out.println(string3);
38
39 System.out.println(string4);
40
```

Problems @ Javadoc Declaration Console Progress

<terminated> OlaMundo [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe

null
a
O rato roeu a roupa do rei de Roma.

1.10 Operadores em Java

Quando queremos executar uma operação, cálculo, rotina, comparação entre outras ações em um algoritmo lançamos mão de um valioso recurso das linguagens de programação que é o operador. Existem vários tipos de operadores de atribuição, aritméticos, relacional, lógico e operador ternário. Neste capítulo veremos quais são e suas aplicações nos diferentes contextos de código que serão as mais amplas possíveis.

1.10.1 Operador de atribuição.

O primeiro e mais importante dos operadores, porque é através dele que damos valores as variáveis ou mudar um valor pré existente, o que chamamos de sobrescrita de valor. É importante ressaltar que a variável ficará a esquerda e o valor a ser atribuído a direita.

```
22 public static void main(String[] args) {
23
24     int var = 2; // a variável do tipo inteiro recebe o valor 2.
25
26     System.out.println("O valor inicial é: "+var);
27
28     var = 10; // o valor da variável é sobrescrito
29
30     System.out.println("O novo valor será: "+var);
31 }
32
```

Problems @ Javadoc Declaration Console Progress

<terminated> OlaMundo [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe

O valor inicial é: 2
O novo valor será: 10

1.10.2. Operadores aritméticos.

Como o próprio nome diz esses operadores representam as operações básicas da matemática como soma (+), subtração (-), multiplicação (*), divisão (/) e etc. Podemos usar mais de um desses operadores em uma sentença matemática para chegarmos ao resultado pretendido, as vezes apenas um deles é suficiente para resolver cálculos simples.

A lei de precedência da matemática também se aplica a programação. Então primeiro serão resolvidos os cálculos entre parênteses, raízes e exponenciais, multiplicação e divisão na ordem que aparecem, soma e subtração na ordem que aparecem, sempre da esquerda para a direita.

Exemplos de operações a serem resolvidas com este recurso são a soma, subtração de dois números, resto de uma divisão, cálculo do IMC (índice de massa corporal) de uma pessoa entre os mais variados.

Operador de módulo % é utilizado quando desejamos saber o resto de uma divisão. Por exemplo 19 % 2 retorna o resultado 1 já que 19 / 2 é igual a 9 e resta 1. 25%7 retorna 4 que é o resto da divisão. É bem simples de entender e muito útil para determinadas situações. Segue abaixo uma tabela com o resumo dos operadores aritméticos vistos nesse tópico.

operador	função	exemplo
+	0	variavel1 + variavel2
-	0	variavel1 - variavel2
*	0	variavel1 * variavel2
/	0	variavel1 / variavel2
%	0	variavel1 % variavel2

```

54 int numero1 = 6, numero2 = 4;
55
56 System.out.println("a soma dos dois números é: "+(numero1+numero2));
57 System.out.println("a subtração dos dois números é: "+(numero1-numero2));
58 System.out.println("a multiplicação dos dois números é: "+numero1*numero2);
59 System.out.println("a divisão dos dois números é: "+numero1/numero2);
60 System.out.println("o resto da divisão dos dois números é: "+numero1%numero2);
61
62
63

```

Problems @ Javadoc Declaration Console Progress

<terminated> OlaMundo [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe

a soma dos dois números é: 10
a subtração dos dois números é: 2
a multiplicação dos dois números é: 24
a divisão dos dois números é: 1
o resto da divisão dos dois números é: 2

1.10.3. Operadores unários.

Operadores unários são de simples entendimento. Eles servem para modificar uma variável ao mesmo tempo que atribuem o novo valor em uma só tacada. Para fazermos uso dos operadores unários lançamos mão dos operadores aritméticos só que dessa vez, ao invés, de termos dois ou mais componentes temos apenas um.

Imagine que temos uma variável chamada **num** que foi atribuído o valor de 10. Quando fazemos **num++** esse valor será incrementado de uma unidade e passará a ser 11. Se fizermos **num--** esse valor será decrementado de uma unidade e voltará ao valor 10, chamamos essa operação de pós incremento e pós decremento.

Também poderíamos fazer **++num** e **--num** que chamamos de pré incremento e pré decremento. A diferença entre o pré e o pós incremento e decremento é que o pré primeiro modifica a variável e depois realiza as operações da linha de código na qual se apresenta, como por exemplo, um print e o pós primeiro realiza o print e depois altera o valor da variável.

O operador unário **!** faz com que os valores booleanos seja invertido, ou seja, um valor **true** se tornará **false** e vice-versa. Será muito utilizado quando estivermos vendo estruturas de decisão.

1.10.4. Operadores de atribuição compostos.

Os operadores de atribuição compostos em termos de aplicabilidade são iguais aos unários de decremento e incremento só que além de apenas somar ou subtrair uma unidade agora podemos usar outros números e outras operações como multiplicação e divisão assim como veremos no exemplo a seguir. Essa é uma ferramenta poderosa para deixar o código mais limpo e simples. Existem cerca de 11 operadores de atribuição compostos, mas veremos somente os mais importantes como **+=**, **-=**, ***=** e **/=**. Lembrando que esses operadores servem quando se quer mudar o valor de uma variável sem ter que ficar repetindo a mesma desnecessariamente.

```
22 public static void main(String[] args) {
23
24     int num = 6;
25
26     num+=10; // 16
27
28     num-=4;  // 12
29
30     num*=10; // 120
31
32     num/=2;  // 60
33
34     System.out.println(num);
35 }
36
```

Problems @ Javadoc Declaration Console Progress
<terminated> OlaMundo [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe
60

1.10.5. Operadores relacionais.

Operadores relacionais vão pegar dois ou mais dados, fazer a comparação entre eles e depois retornar um valor booleano **true** ou **false**. Começemos com

`==` operador relacional igualdade

`!=` operador relacional desigualdade

Esses operadores fazem a comparação entre os valores de duas variáveis e verifica se eles são iguais ou diferentes. Muita atenção o operador de igualdade são dois símbolos de `=`, muito cuidado na hora do código para não confundir e errar. Abaixo temos a tabela verdadeira desses dois operadores.

Operador	igual	diferente
<code>num == num2</code>	0	0
<code>num != num2</code>	0	0

Não podemos usar os comparadores para comparar duas variáveis de tipos primitivos diferentes somente se forem os tipos numéricos, `int`, `float` e `double` entre si, porém `char` e `int`, `String` e `char` não é uma operação que seja possível de ser realizada e retornará um erro em seu código.

`>` maior que

`<` menor que

`>=` maior ou igual que

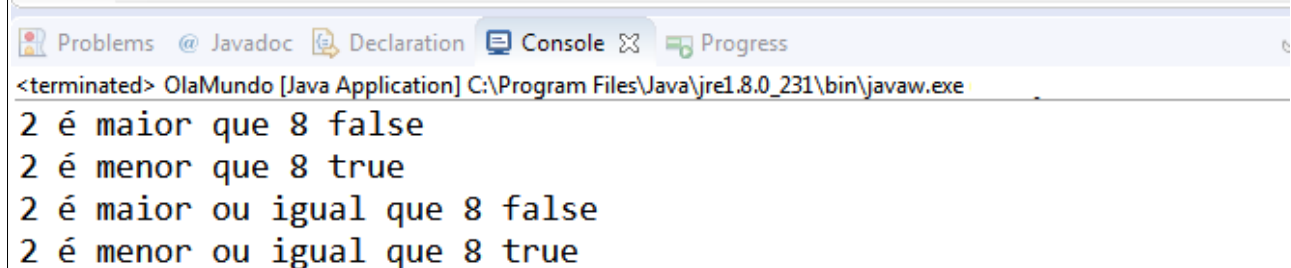
`<=` menor ou igual que

Assim como a igualdade e a desigualdade esses operadores vão fazer a comparação entre duas variáveis e retornar valores booleanos de verdadeiro ou falso. São amplamente utilizados nas estruturas de decisão, como veremos nos próximos capítulos. Para quem tem dificuldade de saber o que é maior e o que é menor é só você pensar o sinal, se verdadeiro, sempre vai apontar para o menor número, maior `>` menor, menor `<` maior. Abaixo temos uma tabela para facilitar o entendimento.

Valores A e B	<code>A > B</code>	<code>A < B</code>	<code>A >= B</code>	<code>A <= B</code>
2 e 2	0	0	0	0
5 e 8	0	0	0	0
10 e 1	true	false	true	false

Veja a figura abaixo a representação da tabela com duas variáveis $a = 2$ e $b = 8$.

```
65 int a=2, b=8;
66
67 System.out.println(a+" é maior que "+b+" "+(a>b));
68 System.out.println(a+" é menor que "+b+" "+(a<b));
69 System.out.println(a+" é maior ou igual que "+b+" "+(a>=b));
70 System.out.println(a+" é menor ou igual que "+b+" "+(a<=b));
71
72
```



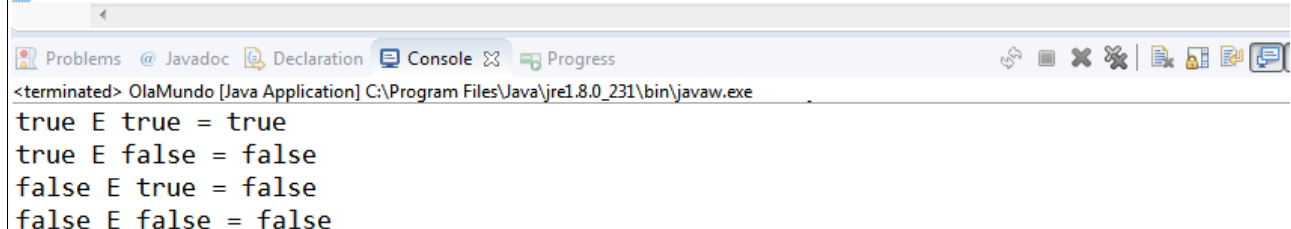
```
<terminated> OlaMundo [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe
2 é maior que 8 false
2 é menor que 8 true
2 é maior ou igual que 8 false
2 é menor ou igual que 8 true
```

1.10.6. Operadores lógicos.

Operadores lógicos vão analisar dois booleanos, diferente do que vimos até agora aonde tratávamos com variáveis, e dependendo do valor destes nós teremos uma tabela verdade. Vamos falar de cada um deles.

&& operador “E” lógico. Só retorna verdadeiro caso todas as proposições lógicas sejam verdadeiras, caso contrário retornará falso. Exemplo: Sua mãe te manda para comprar pão e leite. Você só será terá sucesso na tarefa se comprar os dois, caso falte algum você levará uma bronca.

```
80 boolean Verdadeiro = true, verdadeiro=true, Falso= false, falso = false;
81
82 System.out.println(Verdadeiro+" E "+verdadeiro+" = "+(Verdadeiro&&verdadeiro));
83 System.out.println(Verdadeiro+" E "+falso+" = "+(Verdadeiro&&falso));
84 System.out.println(falso+" E "+verdadeiro+" = "+(falso&&verdadeiro));
85 System.out.println(Falso+" E "+falso+" = "+(Falso&&falso));
86
87
```



```
<terminated> OlaMundo [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe
true E true = true
true E false = false
false E true = false
false E false = false
```

|| operador “OU” lógico. O “ou” retornará verdadeiro se qualquer uma, ou todas as proposições lógicas forem verdadeiras. O único caso no qual o “ou” retornará falso é aquele no qual todas as proposições são falsas. Exemplo: Dessa vez sua tarefa é ir na mercearia e trazer dois shampoos da marca A ou B, ou você pode trazer um de cada. Qualquer arranjo será aceito. Sua tarefa só falhará, caso não traga nenhum para casa.

```
80 boolean Verdadeiro = true, verdadeiro=true, Falso= false, falso = false;
81
82 System.out.println(Verdadeiro+" OU "+verdadeiro+" = "+(Verdadeiro||verdadeiro));
83 System.out.println(Verdadeiro+" OU "+falso+" = "+(Verdadeiro||falso));
84 System.out.println(falso+" OU "+verdadeiro+" = "+(falso||verdadeiro));
85 System.out.println(Falso+" OU "+falso+" = "+(Falso||falso));
86
87
```

<terminated> OlaMundo [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe

true OU true = true
true OU false = true
false OU true = true
false OU false = false

operador “OU EXCLUSIVO” lógico. O ou exclusivo retornará falso, casos dois ou mais proposições lógicas forem verdadeiras, ou se todas forem falsas. O resultado verdadeiro só será obtido caso apenas uma entre todas as proposições forem verdadeiras, por isso o nome ou exclusivo. Exemplo: Sua tarefa agora é comprar frango, ou carne, caso compre os dois levará uma bronca porque um deles vai estragar. Então só terá sucesso se voltar para casa com apenas uma das duas opções.

```
80 boolean Verdadeiro = true, verdadeiro=true, Falso= false, falso = false;
81
82 System.out.println(Verdadeiro+" XOR "+verdadeiro+" = "+(Verdadeiro^verdadeiro));
83 System.out.println(Verdadeiro+" XOR "+falso+" = "+(Verdadeiro^falso));
84 System.out.println(falso+" XOR "+verdadeiro+" = "+(falso^verdadeiro));
85 System.out.println(Falso+" XOR "+falso+" = "+(Falso^falso));
86 System.out.println("Bônus false XOR true XOR true = "+(falso^Verdadeiro^verdadeiro));
87
```

<terminated> OlaMundo [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe

true XOR true = false
true XOR false = true
false XOR true = true
false XOR false = false
Bônus false XOR true XOR true = false

1.10.7. Operador ternário.

Esse operador em específico é dominado por qualquer pessoa que tenha feito um curso básico de Excel ou Calc. Operador terciário é utilizado quando temos uma proposição lógica e queremos um determinado resultado para o caso a resposta seja verdadeira e outro para caso seja falsa. Como os operadores unários e de atribuição

composta é uma ótima ferramenta a ser utilizada para enxugar o código e deixá-lo fácil de ler quando precisamos de uma estrutura de decisão simples. Vejamos um exemplo na figura abaixo:

```
89 //criação das variáveis do tipo int
90 int num = 7, num2=9;
91
92 // uma variável do tipo String vai receber o resultado do operador ternário.
93 String resultado = num>num2?"sete é maior do que nove":"sete é menor do que nove.";
94
95 //impressão do resultado
96 System.out.println(resultado);
97
```

<terminated> OlaMundo [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe
sete é menor do que nove.

Vamos analisar apenas o operador ternário. Na primeira parte temos a proposição lógica que vai pegar num e comparar se é maior do que num2 note que uma parte muito importante é o ponto de interrogação após o teste lógico. Na segunda parte, após o ponto de interrogação temos o valor assumido pelo operador caso o teste retorne verdadeiro e separamos do outro valor por dois pontos. Ficando a estrutura da seguinte maneira.

(teste lógico) ? (resultado para verdadeiro) : (resultado para falso);

Vamos fazer um outro exemplo para fixar melhor o aprendizado. Dessa vez usaremos o exemplo clássico do aluno e sua aprovação.

```
90 double nota = 8;
91
92 // uma variável do tipo String vai receber o resultado do operador ternário.
93 String resultado = nota>=7.0?"aprovado":"de recuperação";
94
95 //impressão do resultado.
96 System.out.println("O aluno está "+resultado);
97
```

<terminated> OlaMundo [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe
O aluno está aprovado

No exemplo abaixo vemos que o teste lógico recebe uma determinada nota e verifica. Caso a nota seja maior ou igual a sete resultado receberá “aprovado”

1.11. Operador Instanceof:

Permite-nos saber se um objeto pertence a uma classe ou não, ou seja se o objeto passa no teste é um. Este operador é usado somente com variáveis de referências de

objetos, e pode ser empregado para verificar se um objeto é do tipo específico. Quando tratamos de tipos, queremos dizer tipo de interface ou classe. Em outras palavras, se o objeto referenciado pela variável à esquerda do operador passaria no teste é um do tipo de interface ou classe do lado direito do operador. A sintaxe do operador instanceof é :

```
if(p instanceof Produto) {
    System.out.println("O objeto p é uma instância da classe Produto.");
}
```

1.12. Operador de concatenação de Strings.

O sinal de adição(+) também pode ser usado para concatenar duas strings. Se um dos valores for uma String, o operador + atuará como um operador de concatenação de Strings. E se os dois valores forem números, o operador + funcionará como o operador de adição. Caso você esteja fazendo uma operação aritmética junto com a concatenação você deve parar as operações através de parênteses assim o compilador conseguirá distinguir um do outro.

1.13. Precedência dos operadores.

A precedência vai dizer exatamente em qual ordem o compilador vai executar os operadores. É muito importante sabê-la para que você possa escrever o código e obter uma resposta coerente com seus objetivos e não acabe colocando os pés pelas mãos na hora de usar essa incrível feature das linguagens de programação. Abaixo segue uma tabela detalhas da sequência de execução dos operadores. Você não precisa decorá-la, mas é sempre bom ter uma por perto para tirar suas dúvidas.

Ordem	Tipo de operador	Precedência
1	Pós fixo	expr++ expr- -
2	Pré fixo, unário	++expr - - expr !
3	Multiplicativo	* / %
4	Aditivo	+ -
5	Relacional	> >= <= <
6	Igualdade	!= ==
7	AND lógico	&&
8	OR lógico	
9	Atribuição	%= /= *= -= += =

1.14 Exercícios.

1. Processo que transforma o código em bytecodes para que possa ser compreendido pela JVM que executará a rotina contida no mesmo.

- a) interpretação
- b) instanciação
- c) compilação
- d) herança
- e) digitação

2. O que é o garbage collector?

- a) é um programa que zera todas as variáveis que serão utilizadas pelo programa para que não tenham nenhum valor indesejado.
- b) é uma rotina do java que escaneia o programa em busca de falhas.
- c) ele verifica a qualidade dos bytecodes
- d) o garbage collector é uma tecnologia exclusiva do java que permite que ele seja portátil para outros dispositivos com sistemas operacionais diferentes.
- e) Sua função é a de varrer a memória de tempos em tempos, liberando automaticamente os blocos que não estão sendo utilizados da memória do computador.

3. É uma máquina imaginária que é implementada através de um software emulador em uma máquina real, provê especificações de plataforma de hardware na qual compila-se todo código de tecnologia Java:

- a) JVM
- b) JRE
- c) JVA
- d) NBA
- e) N.R.A.

4. Qual dessas não é uma das qualidades do java?

- a) multithreading
- b) portátil
- c) garbage collector

d) fracamente tipado

e) processamento distribuído

5. É o operador aritmético que é usado quando queremos retornar o resto da divisão entre dois números.

a) +

b) %

c) \$

d) -

e) *

6. Qual das variáveis abaixo não tem uma nomenclatura válida de acordo com os padrões da linguagem Java?

a) numeroComposto

b) carro_3

c) _nome

d) primeiro nome

e) sinal

7. A IDE mais voltada para o aprendizado acadêmico do paradigma de orientação a objetos e também é utilizado para projetos de pequeno porte. Apesar de ser desenvolvido para os iniciantes aprenderem a programar em Java, muitos programadores experientes preferem usá-la já que permite ter uma visão e controle sobre os objetos e classes assim como sua estrutura.

a) BlueJ

b) IntelliJ

c) Eclipse

d) Netbeans

e) JDeveloper

8. Complete a tabela.

tipo	tamanho	mínimo	máximo	wrapper
boolean				
char	16 bit			
byte				
short				
int		-2^{31}		
long				
float				Float
double			1.79769313486231570E+308	

9. Mostre na mesma linha as variáveis int horas=8, float taxa = 9.4 e char turno = 'N'. Usando os seguintes comandos.

- a) System.out.println
- b) System.out.print
- c) System.out.printf

10. Qual o operador que usamos para simular uma estrutura de decisão simples, composto por: um teste lógico, um retorno para resposta verdadeira, um retorno para uma resposta falsa.

- a) operador de atribuição
- b) operador ternário
- c) operador lógico
- d) operador relacional
- e) operador aritmético

Capítulo 2. Controle de fluxo.

As estruturas de controle definem a **sequência de execução das instruções**. Não é possível implementar um algoritmo que não seja trivial em uma linguagem de programação que não tenha um conjunto mínimo de estruturas de controle. As estruturas de controle podem ser divididas em *seleção*, *repetição* e *sequência*. A sequência é simplesmente definida pela **execução sequencial dos comandos, de cima para baixo**. O controle de fluxo é essencial em qualquer linguagem de programação pois através dele que o programa executa a rotina para o qual foi criado assim como define um programa como imperativo e não declarativo.

Cada programa é feito com um propósito: contar o tempo, classificar produtos, efetuar uma venda, calcular média e várias outras funcionalidades. Para que o programa execute a sua função usaremos as estruturas de controle. Começaremos pelas estruturas de decisão.

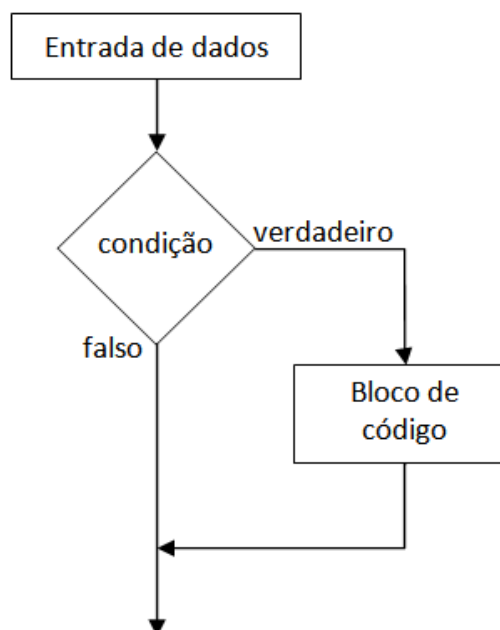
2.1. Estruturas de decisão.

Durante o nosso dia, desde a hora que acordamos até a hora de ir dormir efetuamos várias decisões como levantar da cama; tomar café da manhã; o que comer no almoço; fazer exercícios; entrar em uma loja, algumas espontâneas, outras nem tanto.

No mundo dos algoritmos decisões também devem ser tomadas a todo o tempo para que o programa realize a tarefa para o qual ele foi desenvolvido. Para fazer esse controle de que caminho ele deve seguir, qual instrução será executada, executamos as estruturas de decisão. Nesse curso aprenderemos recursividade junto com **IF-ELSE** e o **SWITCH**.

2.1.1. Estrutura de decisão if.

A estrutura de decisão if é usada para verificar o teste lógico, que estará após a palavra reservada **if** e entre parênteses, e caso esse teste retorne verdadeiro ele executa um bloco de código. Observação: Se a instrução contida dentro do bloco de código for um só comando você não precisará envolvê-lo em chaves como é mostrado na figura abaixo, pois o compilador entenderá que a próxima instrução é a que deve ser executada, caso o teste retorne o valor verdadeiro.



```
boolean campanha = true;

if(campanha) {
    System.out.println("Ding dong!");
}
```

```
String diaDaSemana="Domingo";

if(diaDaSemana == "Sábado" || diaDaSemana == "Domingo")
    System.out.println("Fim de semana");
```

Preste bastante atenção na próxima figura. Usamos o operador unário “!” para inverter o valor lógico da variável e do teste lógico. A variável `revisaoMensal` recebe `true`, porém `esseMes` não foi feita a revisão mensal. Logo a variável `esseMes` recebe `!revisaoMensal` tendo o valor de falso.

Analisemos agora o teste lógico que tem entre parênteses “`!esseMes`”, isso significa que o `if` vai monitorar se esse valor é falso, caso seja ele irá imprimir a mensagem “Não poderá viajar até que seja feita a revisão mensal”. Isso porque o dono do carro é muito precavido.

```
boolean revisaoMensal = true;

boolean esseMes = !revisaoMensal;

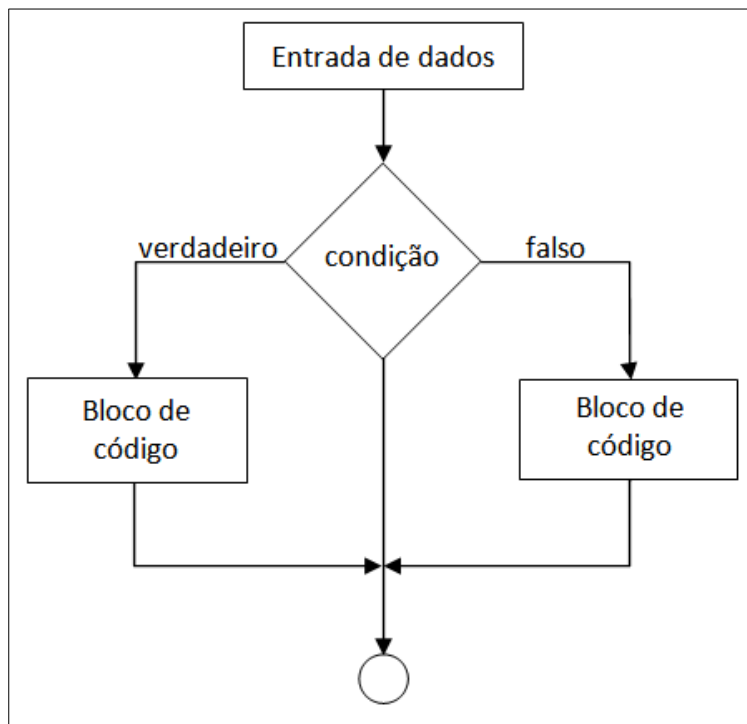
if(!esseMes)
    System.out.println("Não poderá viajar até que seja feita a revisão mensal");
```

2.1.1. Estrutura de decisão if-else

O **if-else** é um avanço da estrutura anterior que tinha apenas um tratamento para o caso verdadeiro. Agora temos o **else** cujo bloco será executado se a condição do **if** não for satisfeita. O fluxograma do **if-else** é mostrado na próxima figura note que na estrutura anterior, caso o programa não entrasse no bloco do **if** ele seguia a execução normalmente agora, se isso acontece, temos um bloco de execução do **else** que será executado no lugar.

O **else** compartilha da mesma particularidade do **if**: caso tenha apenas um comando para executar, você não precisa colocar as chaves para envolver o bloco de

código, pois o compilador entenderá que a próxima linha a ser executada pertence ao else, dessa maneira o código ficará bem mais limpo e simples para entender. Observação: você pode ter apenas o if, porém não pode ter um else desacompanhado.



```
int idade = 18;

if(idade>=18)

    System.out.println("Você pode tirar a habilitação.");

else

    System.out.println("Você não pode tirar a habilitação.");
```

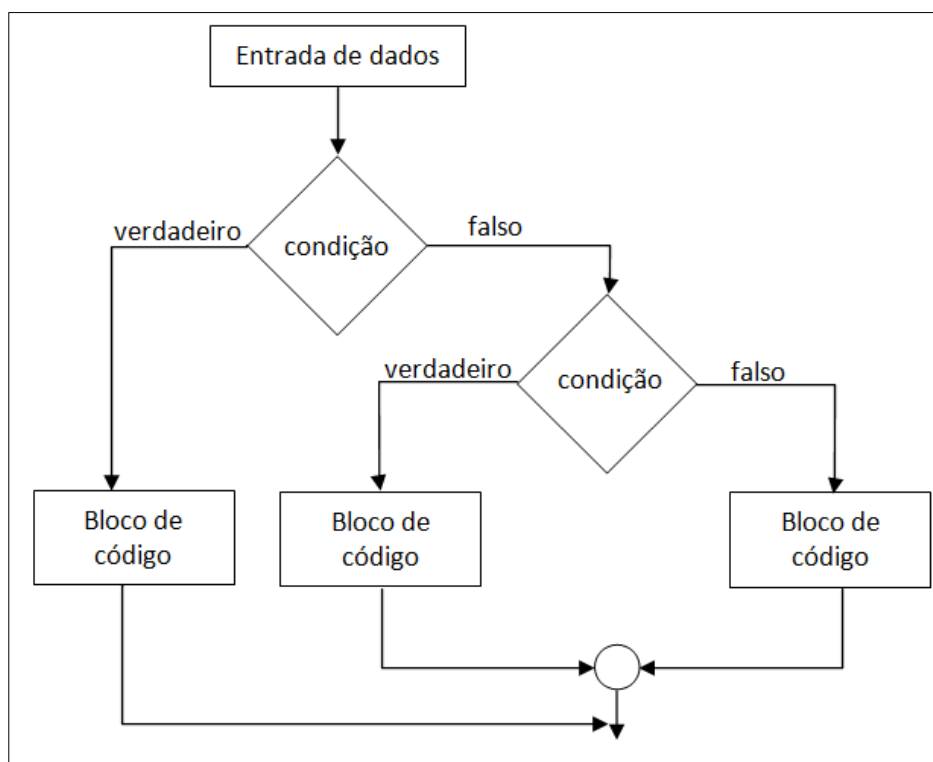
2.1.2. Estrutura de decisão if-else-if

Quando temos mais de um teste lógico que não pode ser simplesmente resolvido com operadores lógicos, o que fazemos, outro if-else? Isso deixaria o código muito carregado, então para resolver esse problema de maneira eficaz e elegante temos os else-if. É uma estrutura que fica entre o if e o else fazendo com que possamos inserir mais um teste lógico e um outro bloco de comandos.

Também segue as mesmas propriedades dos seus antecessores quanto ao não precisar de chaves caso só aja um comando a executar e dentro do teste lógico pode

conter operadores relacionais, lógicos e unários para a obtenção do resultado. Tenha muito cuidado de quando fizer um if-else-if de cobrir todas as possibilidades pretendidas para não haver erros de lógica e caso não seja nenhum deles.

O seu código funcionará perfeitamente só com if-else e com o else-if, porém é sempre uma boa prática de programação colocar um else para os casos gerais que não se enquadrem nas proposições lógicas. Na imagem teremos um efeito cascata todos os **ifs** serão executados um por um até que um deles retorne o valor verdadeiro ou caia no else, caso um deles retorne verdadeiro seu bloco de comandos é executado, logo após a execução da estrutura if é terminada.



- Vamos usar e abusar do bom e velho exemplo do aluno e sua nota. No código abaixo temos a declaração da nota, declarada inválida de propósito.
- O primeiro if verifica se a nota é maior do que 7 e menor e igual a 10 (necessário para restringir a nota máxima em 10), caso seja verdade o aluno passa por nota.
- O primeiro else-if analisa se a nota é menor do que 7 e maior ou igual a 3, para que o aluno possa ir para a recuperação.
- O segundo else-if verifica se a nota é menor do que 3 e maior ou igual a 0 (nota mínima) se for verdade então o aluno foi reprovado por nota.
- O else serve para fechar com chave de ouro. Como todas as opções anteriores cobrem todas as possibilidades de notas válidas (de 0 a 10), caso seja passado

qualquer outro valor numérico fora desse intervalo o else passará uma mensagem de nota inválida.

```
double nota=15;

if(nota>=7 && nota<=10)

    System.out.println("O aluno passou de ano.");

else if(nota<7 && nota>=3)

    System.out.println("O aluno está de recuperação.");

else if(nota<3 && nota>=0)

    System.out.println("o aluno foi reprovado por nota.");

else

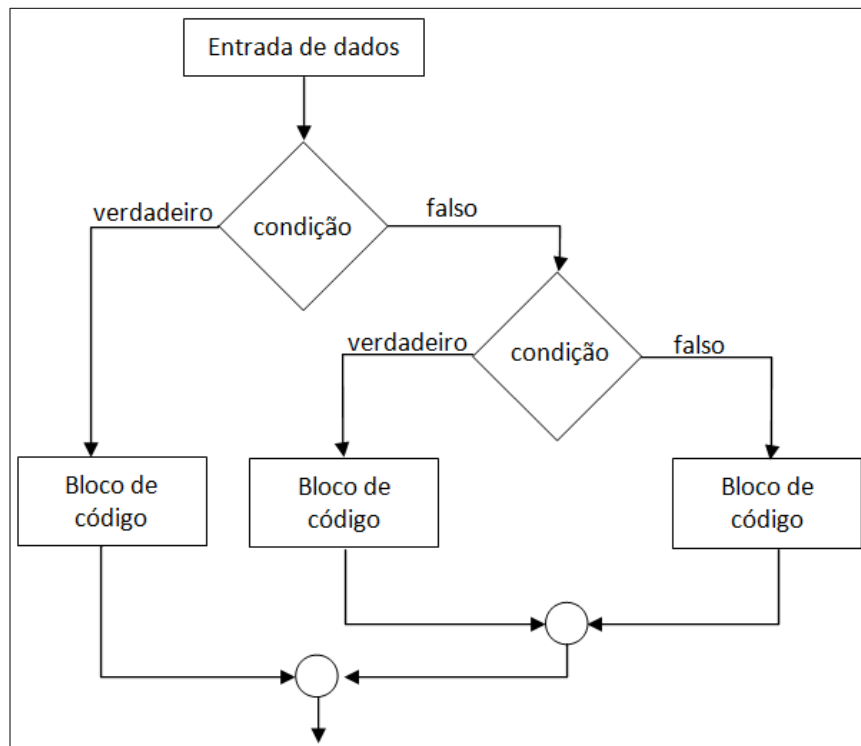
    System.out.println("nota inválida.");
```

2.1.3. Estrutura de decisão if-else aninhado.

Chamamos de if-else aninhado quando temos uma condição dentro de uma condição. O diagrama de fluxo do if-else aninhado é bem semelhante ao do if-else-if a diferença é que, ao invés, de termos mais um teste lógico principal temos um dentro de outro, logo ele só será executado se o código entrar naquele bloco de comandos.

Continuemos com o nosso exemplo super didático do nosso aluno em sua busca pela aprovação. Dessa vez o aluno teve uma nota 6, o que foi suficiente para fazer a recuperação. Dentro da recuperação temos uma análise da nota da recuperação, se tirar um 7 ele passará, caso contrário, ficará reprovado. Levando em consideração esse exemplo podemos observar que temos uma estrutura de decisão dentro da outra referente a ele ter a sua segunda nota analisada para só então o programa calcular qual será o resultado final.

Apesar dessas estruturas serem muito úteis, não é recomendado usar de maneira indiscriminada. Os riscos de uma má utilização do IF é a perda da linearidade do código. Quanto mais IFs tivermos será mais difícil de acompanhar o fluxo do programa, isso tornará mais árdua a tarefa de dar manutenção e até mesmo de ser lido por terceiros. Estamos falando aqui não de banir o uso dessa estrutura, mas sim do uso racional e consciente para que ela seja um aliado e não um empecilho.



```
double nota=6.0, notaRecuperacao=7.8;

if(nota>=7 && nota<=10)

    System.out.println("O aluno passou de ano.");

else if(nota<7 && nota>=3) {

    System.out.println("O aluno está de recuperação.");

    if(notaRecuperacao>=7)

        System.out.println("O aluno passou de ano na recuperação.");

    else

        System.out.println("o aluno foi reprovado por nota na recuperação");

}

else if(nota<3 && nota>=0)

    System.out.println("o aluno foi reprovado por nota.");

else

    System.out.println("nota inválida.");
```

2.1.4. Estrutura de decisão SWITCH

Quando temos a análise de uma variável (byte, short, int, char, String) para a partir dela decidirmos que rumo nosso programa tomará podemos usar a estrutura de decisão **SWITCH**. Que é a mais indicada para a construção de menus e afins. O seu diagrama é

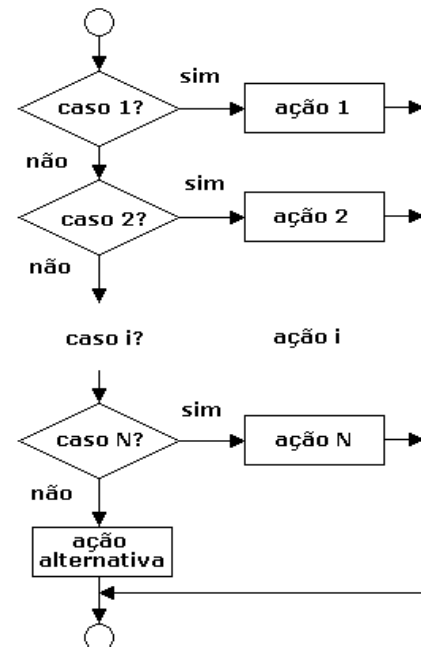
bem simples. Temos a análise de uma variável e a comparação com cada um dos casos contidos nela. Se o programa encontrar compatibilidade com algum caso a análise termina e o mesmo será executado. Se não encontrar nenhuma ocorrência então a opção padrão (DEFAULT) será executada. Note que ao fim de cada caso usamos a palavra reservada `break` para que o `switch` seja encerrado, uma vez que na ausência dela todos os casos serão executados até chegar ao default.

```

24 public static void main(String[] args) {
25
26
27
28     int escolha = 1;
29
30     switch(escolha) {
31
32         case 1: System.out.println("Novo Jogo"); break;
33         case 2: System.out.println("Continuar"); break;
34         case 3: System.out.println("Apagar gravação"); break;
35         case 4: System.out.println("Configurações"); break;
36         default: System.out.println("Opção inválida");
37     }
38 }
39
40
41
42
43

```

Problems @ Javadoc Declaration Console Progress
 <terminated> OlaMundo [Java Application] C:\Program Files\Java\jre1.8.0_281\bin\javaw.exe
 Novo Jogo



A observação que fizemos sobre o IF também é válida para o `switch`. Deve-se usá-lo da maneira correta, ou o código ficará difícil de manter e acompanhar. No próximo exemplo nós temos um código no qual a variável de decisão é o carácter. Para definir qual é o tipo de cadastro na corrida e qual será o brinde que o participante receberá.

```

28     char cadastroCategoria = 'c';
29
30     switch(cadastroCategoria) {
31
32         case 'a': System.out.println("mochila, camisa do evento, boné , squeeze, medalha, chip de rastreo"); break;
33         case 'b': System.out.println("camisa do evento, boné , squeeze, medalha, chip de rastreo"); break;
34         case 'c': System.out.println("boné , squeeze, medalha, chip de rastreo"); break;
35         case 'd': System.out.println("squeeze, medalha, chip de rastreo"); break;
36         case 'e': System.out.println("medalha, chip de rastreo"); break;
37         default: System.out.println("cadastro inválido");
38     }
39
40
41
42
43
44
45

```

Problems @ Javadoc Declaration Console Progress
 <terminated> OlaMundo [Java Application] C:\Program Files\Java\jre1.8.0_281\bin\javaw.exe
 boné , squeeze, medalha, chip de rastreo

2.2. Estruturas de repetição.

As vezes no nosso algoritmo nos deparamos com situações onde precisamos repetir algum comando, várias vezes, repetidamente. Então você, como programador, tem duas escolhas: escrever o mesmo comando, de novo, de novo e de novo, ou usar uma estrutura que repetirá esse comando quantas vezes for preciso. Essas são as estruturas de repetição, também conhecidas como laços (loops). Elas vão executar um comando até que atinja o número de repetições desejado (por contador), ou até que receba algum sinal dizendo que não é mais necessário executar aquele comando (flag). Vamos começar vendo recursividade. Para entender um pouco mais das outras.

2.2.1 Recursividade.

Recursividade é o mecanismo básico para repetições na programação funcional que consiste em uma função, ou objeto que chama a si mesmo durante a execução da sua rotina por um número limitado de vezes.

```
public static int soma(int sum ) {  
    if(sum==1) {  
        return 1;  
    }  
    return sum+soma(sum-1);  
}
```

Nosso primeiro exemplo de recursividade é do somatório de N números. É uma função que recebe um número inteiro e verifica se o número que recebeu é igual ou menor a 1 ela para de executar retornando 1, caso não seja ela retorna a si mesma decrementando o número em uma unidade.

```
public static int fatorial(int fat) {  
    if(fat==0) return 1;  
  
    return fat * fatorial(fat-1);  
}
```

Exemplo de uma função recursiva que calcula o fatorial de um número, retornando a multiplicação dele pela chamada da função.

```
public static int fibonacci(int fib) {  
    if (fib == 1 || fib == 0) return 1;  
  
    return fibonacci(fib - 1) + fibonacci(fib - 2);  
}
```

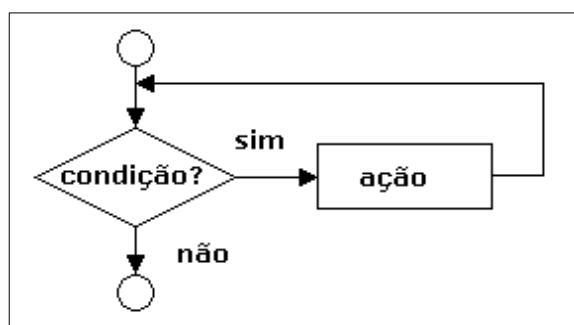
O próximo exemplo retorna o enésimo termo da sequência de fibonacci, onde o retorno é a soma de um número com o seu antecessor, até esse número zerar.

```
public static int exponencial(int base, int exp) {  
    if(exp==0) return 1;  
  
    return base * exponencial(base, exp-1);  
}
```

O último exemplo de recursividade faz o cálculo de um número elevado a um determinado expoente.

2.2.2. Estrutura de repetição WHILE.

O diagrama de fluxo do while é bem simples. Ele vai primeiro analisa uma expressão, se ela for verdadeira ele executa o bloco de código contido nele e retorna para o início, fazendo isso até que o teste lógico retorne falso, quando isso acontecer o laço será encerrado. O maior cuidado que devemos ter com laços, é deixar bem claro o número de repetições ou a condição de término do mesmo, caso contrário nós teremos um loop infinito fazendo o programar executar indefinidamente.



Nosso primeiro exemplo de while é um programa que soma todas as frações de 1 até 1/100. Imagina o trabalho que daria para fazer soma por soma. Demoraria muito tempo para fazer uma expressão $1 + 1/2 + 1/3 + 1/4 + 1/5 + 1/6 \dots 1/100$. Tal expressão seria impraticável por ser muito extensa e fácil de errar enquanto é digitada. Com as estruturas de repetição escrevemos a soma uma vez e repetimos cem vezes.

Antes de mais nada criamos duas variáveis o contador e o resultado. Depois na nossa estrutura **WHILE** dizemos para repetir o comando até o contador chegar no cem. Dentro do bloco de execução repare que usamos **resultado += ((double) 1/contador);** explicaremos cada um separadamente.

A expressão **resultado += x** é a forma mais simplificada de escrever **resultado = resultado + x**. Como contador é um inteiro então a divisão $1 / \text{contador}$ tende a retornar um valor inteiro por isso colocamos o (double) esse recurso é chamado de **casting** e sua

função é transformar esse resultado de inteiro para um número de ponto flutuante do tipo double. Logo após temos uma parte vital do programa que é incrementar o contador com **contador++** sem esse trecho do código o contador não mudaria de valor e entraríamos em um loop infinito. Por último temos a impressão do resultado.

```
7
8 public class Enquanto {
9
10 public static void main(String[] args) {
11
12     int contador=1;
13     double resultado=0;
14
15     while(contador<=100) {
16
17         resultado+= ((double)1/contador);
18
19         contador++;
20
21     }
22     System.out.println(resultado);
23 }
24
25 }
```

Problems @ Javadoc Declaration Console Pro

<terminated> Enquanto [Java Application] C:\Program Files\Java\jre1.8.0_281\bin\javaw.exe
5.187377517639621

```
28 Scanner leitor = new Scanner(System.in);
29
30 boolean flag = true;
31
32 while(flag) {
33
34     System.out.println("Digite qualquer número para elevar ao quadrado e zero para finalizar o programa.");
35
36     int numero = leitor.nextInt();
37
38     if(numero==0)
39         flag = false;
40     else
41         System.out.println(numero*numero);
42
43 }
44
45 System.out.println("Programa encerrado.");
46
```

Problems @ Javadoc Declaration Console Progress

<terminated> Enquanto [Java Application] C:\Program Files\Java\jre1.8.0_281\bin\javaw.exe

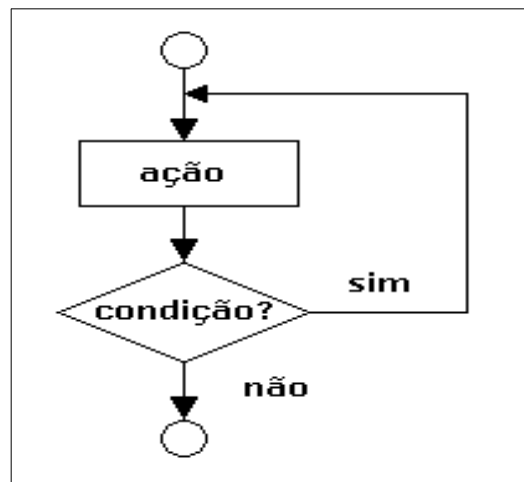
16
256
Digite qualquer número para elevar ao quadrado e zero para finalizar o programa.
9
81
Digite qualquer número para elevar ao quadrado e zero para finalizar o programa.
0
Programa encerrado.

No próximo exemplo temos um programa que pede que o usuário digite um número, verifica se é igual a zero (condição de parada) caso seja ele seta a flag para false

finalizando o loop. Caso não seja ele imprime o quadrado do número digitado. Note que usamos uma instância do objeto **Scan** para receber a informação do usuário.

2.2.3. Estrutura de repetição DO WHILE.

A expressão **do while** é semelhante em todos os aspectos ao **while** exceto por um detalhe. O laço vai executar, pelo menos uma vez, suas instruções, independente do valor do seu teste lógico e só depois faz a verificação para saber se o laço continuará sendo executado, enquanto o **while** só vai ser executado se sua condição for verdadeira. Qual a aplicabilidade para tal estrutura? Imagine um sistema que ao ser ligado, mesmo que não tenha nenhum problema aparente, faça uma checagem de segurança preventiva. Logo abaixo temos o diagrama de fluxo **do do while**.



O nosso exemplo com o **do while** é meramente figurativo. Ele não faz escaneamento nenhum, é só para demonstrar que mesmo que a flag erro seja falsa, e a condição verificando se ela é verdadeira, nosso laço vai executar pelo menos uma vez. Depois da primeira execução a estrutura vai identificar a condição como falsa e vai parar sua execução. Também mostramos os dois diagramas de fluxo para vocês verem de maneira mais clara a diferença entre os dois.

```
15 public static void main(String[] args) {
16
17
18     boolean erro=false;
19
20     do {
21         System.out.printf("Escaneando em busca de erros\n");
22         System.out.printf("...\n...\n...\n");
23         System.out.printf("Diagnóstico terminado erro= %b",erro);
24     }while(erro);
25 }
```

Problems @ Javadoc Declaration Console Progress

<terminated> Enquanto [Java Application] C:\Program Files\Java\jre1.8.0_281\bin\javaw.exe

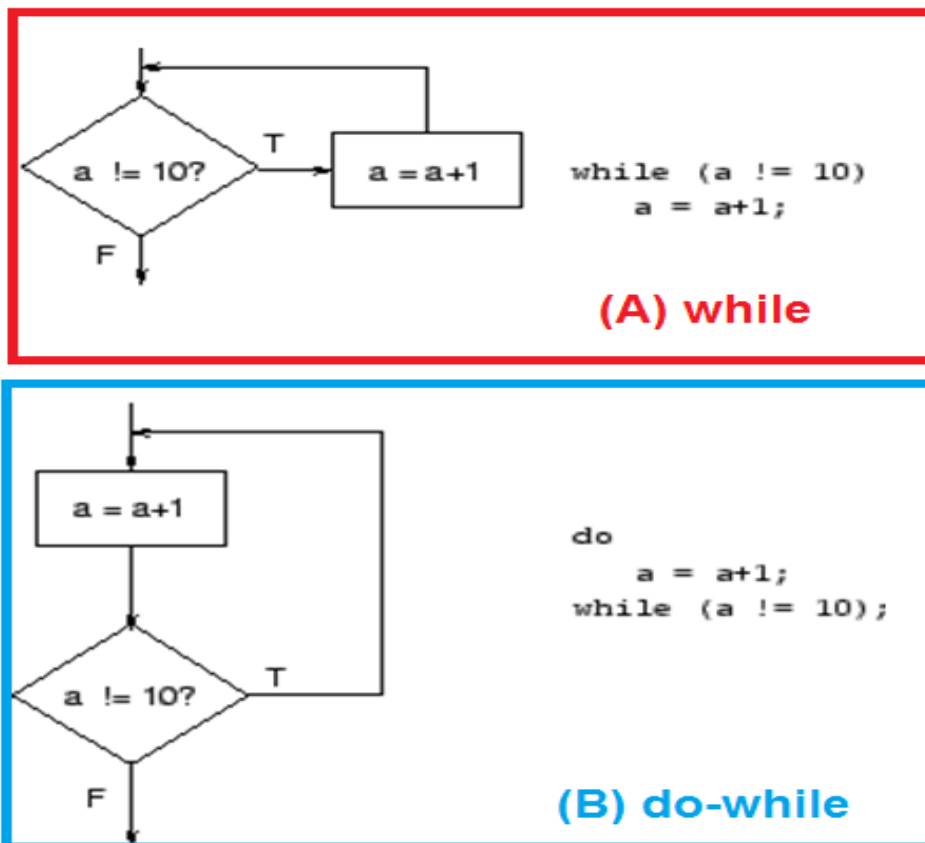
Escaneando em busca de erros

...

...

...

Diagnóstico terminado erro= false



2.2.4. Estrutura de repetição FOR.

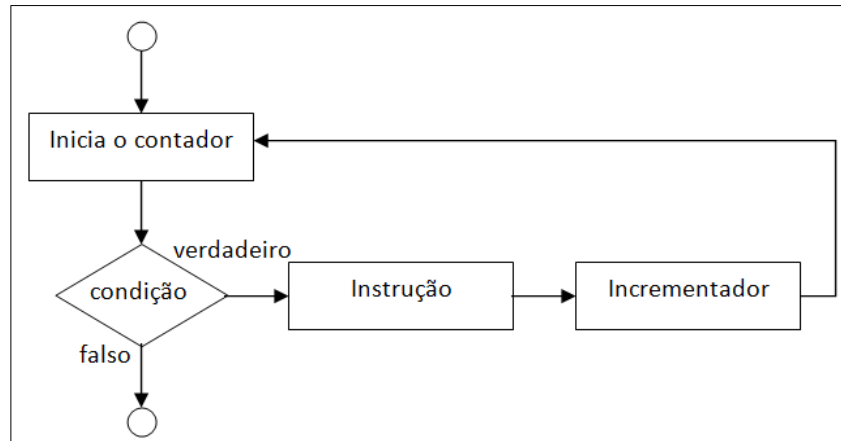
Enquanto no **while** precisávamos ficar declarando dentro do corpo a incrementação do contador, ou mudança da flag, tínhamos que declarar as variáveis de controle e só cabia a estrutura o teste lógico para o controle do laço. No **for** já temos todos esses dados declarados em apenas uma linha, ou seja, bem mais fácil de acompanhar as condições sobre as quais o laço está sujeito.

Essa organização do **for** faz dele o candidato perfeito para trabalharmos com Arrays, tanto é que foi criado o **forEach** só para essa finalidade.

Antes de falarmos mais sobre o **for**. Observe apenas a próxima figura para ver a diferença de verbosidade entre ele e o **while** e compare os dois quanto a clareza e simplicidade de código.

<pre>int x = 0; while(x<=100) { System.out.println(x); x+=2; }</pre>	<pre>for(int x=0; x<=100; x+=2) { System.out.println(x); }</pre>
---	---

Os dois códigos têm a mesma função: imprimir os números pares de 0 a 100. Como podem ver o segundo consegue entregar em apenas três linhas de código o que o primeiro precisou de cinco. Parece pouca coisa, mas em um código cada linha a menos faz diferença na hora de dar manutenção ou ler o mesmo. Verbosidade em excesso apenas deixa o seu código menos funcional. Agora que vimos a diferença, vamos conhecer mais a fundo o **FOR**.



Como podemos ver na figura abaixo na declaração do for já temos a inicialização da variável, a condição de parada e o incremento, tudo junto e logo após temos o corpo onde ficarão os comandos a serem executados durante o laço. Com num inicializado em 0, executando enquanto ele for menor do que 3 e incrementando de 1 a cada volta, teremos três execuções.

```
1 package modulo2;
2
3 public class ForClasse {
4
5     public static void main(String[] args) {
6
7         for(int num = 0; num <3; num++) {
8             System.out.println("Não há lugar como o nosso lar.");
9         }
10
11     }
12 }
```

Problems @ Javadoc Declaration Console Progress

<terminated> ForClasse [Java Application] C:\Program Files\Java\jre1.8.0_281\bin\javaw.exe

Não há lugar como o nosso lar.
Não há lugar como o nosso lar.
Não há lugar como o nosso lar.

O for melhorado foi implementado a partir do Java 5 para servir como facilitador da exploração de Arrays, por enquanto veremos apenas essa versão. Com o lançamento do Java 8 tivemos as lambdas expressions para adicionar mais algumas funcionalidades como o map e filter, veremos mais durante o capítulo de Arrays. Note que esse for

melhorado basicamente pega um array e o quebra em várias partes menores que você especifica no cabeçalho do for e que pode tratar como quiser.

2.3.5 Exercícios propostos.

1) Responda com V para verdadeiro e F para falso.

- () O if é uma estrutura de repetição.
- () O while só executa seu bloco interno caso a condição nele seja verdadeira.
- () O do-while irá executar pelo menos uma vez independente da sua condição ser verdadeira ou falsa.
- () Não existe a necessidade do break em uma estrutura switch case.
- () No cabeçalho do for podemos inicializar a variável, ter a condição do laço e o incremento em um único lugar.

2) Faça um laço que imprima os primeiros 10 números múltiplos de 8.

3) É a estrutura de decisão que analisa uma determinada variável e de acordo com o seu resultado executa algum bloco de código específico.

- a) for
- b) if else
- c) while
- d) do while
- e) switch

4) O que acontece caso um laço não possa alcançar a sua condição de parada?

- a) o programa não roda.
- b) laço infinito.
- c) a VM desliga automaticamente a aplicação.
- d) acontece um erro.
- e) nada acontece.

5) Supondo que a estrutura switch case esteja implementada de maneira correta e completa. Ao analisar uma variável o switch não encontrou nenhuma combinação com os seus cases, nesse caso será executado o:

- a) break
- b) default
- c) o último case.
- d) N.R.A.

6) Qual dessas estruturas é a mais indicada para o manejo de arrays?

- a) while
- b) do while
- c) if else
- d) switch
- e) for

7) Faça um programa que resolva uma equação do 2º grau.

8) Faça um programa que imprima os 10 primeiros números da sequência de Fibonacci

9) Desenvolva um programa que receba um número e diga se ele é um quadrado perfeito, ou não.

10) Com os seus conhecimentos adquiridos até agora faça um programa que receba dois números e verifique se eles são primos entre si.

3. Programação orientada a objetos. POO

Java é uma programação procedural e orientada a objeto. Programação procedural é aquela na qual escrevemos um algoritmo que resolva um determinado problema apontando todo o passo a passo e procedimentos percorridos pelo mesmo até a resolução desse problema. A programação Orientada a Objeto traz consigo várias vantagens como prover uma estrutura de programa mais clara e estruturada. O programa fica menos verboso com a possibilidade de reutilizar os objetos. Com uma boa estrutura temos um programa mais rápido e fácil de executar. Vamos aprender os componentes da programação orientada a objeto.

```
6 public class ForClasse {
7
8     public static void main(String[] args) {
9
10
11         String menu[] = new String[6];
12
13
14         menu[0]= "pizza";
15         menu[1]= "bolo";
16         menu[2]= "pastel";
17         menu[3]= "esfirra";
18         menu[4]= "hamburguer";
19
20         for(String lanches : menu ) {
21             System.out.println(lanches);
22         }
23
24 }
```

Problems @ Javadoc Declaration Console

<terminated> ForClasse [Java Application] C:\Program Files\Java\jr

pizza
bolo
pastel
esfirra
hamburguer
null

3.1. Componentes da Programação Orientada a Objetos.

Os objetos são essenciais para a compreensão da tecnologia orientada a objetos. No nosso dia a dia você encontrará muitos exemplos de objetos do mundo real: seu guarda-roupas, sua geladeira, seu animal de estimação, seu carro. Os objetos do mundo real compartilham duas características: todos eles têm atributos (características) e métodos (ações). O guarda-roupas tem características como número de portas, gavetas e ações como guardar roupas, abrir e fechar portas e gavetas. A geladeira também tem como de características, marca, cor, consumo e métodos como congelar, esfriar, soltar água e gelo, conservar alimentos.

Uma ótima maneira de se acostumar aos objetos é notar aqueles que estão ao seu redor no cotidiano, observe e imagine quais seriam os seus atributos, e quais as ações que ele executa. Um exemplo disso é um carro. Imagine seus atributos como: número de portas, cor, marca, motor. Agora imagine quais as ações que ele executa, como movimentar-se, aumentar a velocidade, diminuir a velocidade, parar, consumir combustível, desligar e ligar o ar-condicionado. Você pode também observar um animal e refletir sobre esses dois aspectos como um gato que tem raça, cor, volume do pelo e realiza ações como: miar, dormir, subir muro, caçar ratos e entre outros. Anote suas observações de maneira organizada e deixe bem claro as suas impressões.

Assim como no mundo real, os programas também são compostos de módulos. Cada um com características específicas e ações que precisam ser executadas. Como

um produto de um supermercado em um sistema. O software deverá guardar os dados dele como nome, marcar, preço quantidade e o registro de código de barras. Olhando por esse lado você consegue perceber a semelhança com um objeto no mundo real?

```
Cachorro c1 = new Cachorro();  
  
c1.nome = "Spark";  
c1.raca = "rafeiro";  
c1.cor = "marrom";  
c1.tamanho = 1.0;  
  
c1.sentar();  
c1.latir();  
c1.passear();  
c1.correr();|
```

Uma classe é um projeto ou protótipo a partir do qual os objetos serão criados dentro do nosso programa. Imagine como um molde, ou planta pelos quais vários objetos serão criados. Quando criamos um objeto a partir de uma classe dizemos que estamos instanciando o objeto. Exemplo claro é uma máquina de produzir sabão. Ela vai seguir determinadas especificações para produzir vários sabões que terão medidas padronizadas, a marca da empresa, e uma embalagem.

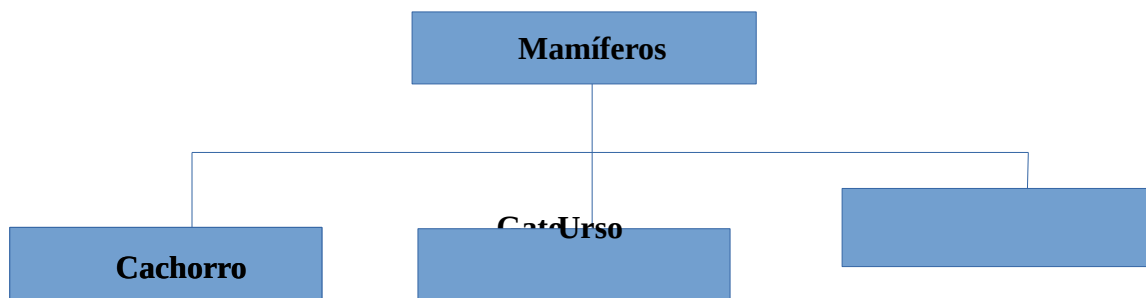
Imagine que mesmo assim essa máquina produzirá vários tipos diferentes: sabão de coco, neutro, de limão e etc e poderão ser também de tamanhos diferentes, pequeno, médio e grande, mas todos eles serão produzidos de acordo com as especificações postas na máquina que seria a classe e os sabões produzidos os objetos criados.

Abaixo vemos a classe cachorro que servirá para produzir vários objetos do tipo cachorro, que apesar de características diferentes como raça, cor do pelo e tamanho, serão todos produzidos a partir do mesmo molde com o mesmo comportamento, ou não (polimorfismo), tais atributos serão criados no decorrer da execução do código.

```
public class Cachorro {  
  
    String nome;  
    String raca;  
    String cor;  
    double tamanho;  
  
    void latir() {  
        System.out.println("au au au");  
    }  
  
    void sentar() {  
        System.out.println("sentado");  
    }  
  
    void passear() {  
        System.out.println("passear");  
    }  
  
    void correr() {  
        System.out.println("correr");  
    }  
  
}
```

A herança fornece um mecanismo poderoso e natural para organizar e estruturar seu software. Diferentes tipos de objetos têm entre si muitas coisas em comum como cães, gatos e ursos ao serem classificados como mamíferos, ou águia, harpia e canário como aves e até joaninha, louva-deus e grilo como insetos.

Notem que todos esses parecem descender de uma só espécie e compartilham de vários traços semelhantes. Para implementar essa característica do mundo real que usamos a herança. Através da herança ao invés de reescrevermos o mesmo código diversas vezes nós podemos criar uma super classe que terá suas características e métodos herdados por outras classes. Como no exemplo abaixo.



Como você já aprendeu, os objetos definem sua interação com o mundo exterior por meio dos métodos que expõem e representam seu comportamento com todos os

outros elemento. Os métodos formam a interface do objeto com o mundo exterior; as teclas do seu teclado são a interface que fazem a comunicação da placa de circuitos com a estrutura de plástico. Você digita e as teclas geram impulsos elétricos que interagem com o processador e retorna o resultado para o monitor. Em sua forma mais comum, uma interface é um grupo de métodos relacionados com corpos vazios.

Um pacote serve para organizar classes e interfaces de maneira lógica através de pastas. Colocar seu código em pacotes facilita o gerenciamento e de grandes projetos.

3.1.1 Classes

Como dito antes, as classes são códigos feitos para, através delas instanciarmos objetos. Nesse capítulo veremos os principais componentes das classes. Vale ressaltar que os nomes das classes sempre devem iniciados com letra maiúscula.

3.1.1.1. Atributos e métodos.

Atributos são as características do objeto. Como exemplo usaremos o objeto Estudante para aprendermos mais afundo todas as peculiaridades da programação orientada a objeto. Para inserirmos atributos dentro de uma classe devemos declarar o tipo do atributo seguido do seu nome.

A classe estudante tem os seguintes atributos:

- cadastro do tipo long
- nome do tipo String
- idade do tipo int
- turma do tipo String
- matriculaAtiva do tipo boolean

```
public class Estudante {  
  
    Long cadastro;  
    String Nome;  
    int idade;  
    String turma;  
    boolean matriculaAtiva;  
  
}
```

3.1.1.2. Modificadores de acesso.

Os modificadores de acesso servem para determinar a visibilidade dos atributos de uma classe, são eles: **private**, **default**, **protected**, **public**.

Private: Os métodos e atributos desse tipo só podem ser acessados dentro da própria classe.

Default: Os métodos e atributos desse tipo podem ser acessados dentro da própria classe e por classe que estejam no mesmo pacote.

Protected: Os métodos e atributos desse tipo podem ser acessados dentro da própria classe, por

```
public class Estudante {  
  
    private long cadastro;  
    protected String Nome;  
    protected int idade;  
    public String turma;  
    public boolean matriculaAtiva;  
  
}
```

classe que estejam dentro do mesmo pacote e em pacotes diferentes, desde que sejam subclasses da mesma.

Public: Pode ser acessado de qualquer parte do projeto.

3.1.1.3 Métodos Construtores

Métodos construtores servem para na instanciação do objeto já setarmos o valor de seus atributos para deixar o código menos verboso ao não precisar chamar levemente os métodos assessores.

```
public class Estudante {  
  
    private long cadastro;  
    private String Nome;  
    private int idade;  
    private String turma;  
    private boolean matriculaAtiva;  
  
    public Estudante(long cadastro, String nome, int idade, String turma) {  
        this.cadastro = cadastro;  
        Nome = nome;  
        this.idade = idade;  
        this.turma = turma;  
        this.matriculaAtiva = true;  
    }  
}
```

O construtor deve ser public, a não se quando aplicado o padrão de projeto singleton, nos parâmetros do mesmo deve haver as variáveis que serão passadas para os atributos, a não ser que a variável seja inicializada junto com o construtor, como é o caso de matriculaAtiva. O construtor também pode conter estruturas de decisão para a inicialização das variáveis.

3.1.1.4 Métodos assessores.

Com os atributos todos privados fica-se impossível de ter acesso aos atributos de fora da classe, no caso acima esse acesso só aconteceria na criação da mesma, porém para resolver este problema nós temos os métodos assessores getters and setters que nos permitem, recuperar e/ou alterar os atributos de fora da classe.

```
public String getTurma() {  
    return turma;  
}  
  
public void setTurma(String turma) {  
    this.turma = turma;  
}  
  
public boolean isMatriculaAtiva() {  
    return matriculaAtiva;  
}  
  
public void setMatriculaAtiva(boolean matriculaAtiva) {  
    this.matriculaAtiva = matriculaAtiva;  
}
```

Na figura acima temos exemplos de métodos assessores dos atributos turma e Matricula ativa.

3.1.1.5 Métodos.

Os métodos são funções aplicadas ao conceito de orientação a objeto. O método tem retorno e parâmetros. Os parâmetros são dados que são recebidos pelo método para serem tratados e o retorno é o valor que será retornado na chamada do método.

```
public void declaracao() {  
    System.out.println("=====Declaração de Matricula=====");  
    System.out.println("O aluno "+this.Nome+" com a matrícula: "+this.cadastro);  
    System.out.println("está matriculado na turma: "+this.turma+" e a situação: "+this.matriculaAtiva);  
    System.out.println("=====");  
}  
  
public double media(double nota1, double nota2) {  
    return (nota1+nota2)/2;  
}  
  
public void boletim(double media) {  
    if(media>=7 && media<=10 ) {  
        System.out.println("O aluno foi aprovado!");  
    }else if(media>=3 && media<7) {  
        System.out.println("O aluno está de recuperação!");  
    }else {  
        System.out.println("O aluno está reprovado");  
    }  
}
```

3.1.1.5.1 Método sem parâmetro e sem retorno.

O método declaração não recebe nenhum parâmetro e não retorna nenhum valor para o programa, neste caso ele só imprime no console a declaração do aluno com alguns dados dele.

```
=====Declaração de Matricula=====  
O aluno José da Silva com a matrícula: 233  
está matriculado na turma: B e a situação: true  
=====
```

3.1.1.5.2. Método com parâmetro e com retorno.

O método média recebe como parâmetro duas notas e calcula a média aritmética entre as duas, tanto os parâmetros quanto o retorno é do tipo double.

```
public class TesteClasses {  
    public static void main(String[] args) {  
        Estudante e1 = new Estudante(233, "José da Silva", 22, "A");  
        e1.setTurma("B");  
        e1.setIdade(19);  
        System.out.println("A média é: "+e1.media(9, 10));  
    }  
}
```

```
<terminated> testeCl:  
A média é: 9.5
```

3.1.1.5.3. Método com parâmetro e sem retorno.

O método boletim recebe a nota e não retorna informação nenhuma, ela apenas verifica o valor da nota e retorna se houve aprovação, recuperação, ou reprovação.

```
public class TesteClasses {  
    public static void main(String[] args) {  
        Estudante e1 = new Estudante(233, "José da Silva", 22, "A");  
        e1.setTurma("B");  
        e1.setIdade(19);  
        e1.boletim(e1.media(10, 4));  
    }  
}
```

```
<terminated> testeClasses.p:  
o aluno foi aprovado!
```

3.1.2 Objetos.

A partir da classe estudante criaremos um objeto e1 passando nele todos os valores no construtor e depois fazendo a modificação através do **set** e recuperando os valores pelo **get**.

Primeiramente vamos criar uma classe de teste. A mesma precisa ter um método **public static void main(String[] args)** dentro dele criando a instancia de um objeto com todos os atributos, depois modificaremos turma e idade e por fim vamos mostrar todos os

atributos do objeto. Se você der um `console.log` no objeto ele vai retornar o endereço de memória no qual o mesmo está guardado.

```
public class TesteClasses {  
    public static void main(String[] args) {  
        Estudante e1 = new Estudante(233, "José da Silva", 22, "A");  
  
        e1.setTurma("B");  
        e1.setIdade(19);  
  
        System.out.println(e1.getCadastro());  
        System.out.println(e1.getNome());  
        System.out.println(e1.getIdade());  
        System.out.println(e1.getTurma());  
    }  
}
```

<terminated> TesteClasses [Java, 233
José da Silva
19
B

Esta é a forma na qual criamos novos objetos a partir de uma classe.

3.2. Herança

Quando temos várias classes que têm atributos e métodos em comum para evitar reescrever o mesmo código, várias vezes, nós podemos fazer com que as classes herdem de uma outra classe usando a palavra reservada `extends`.

```
public class Pessoa {  
    private long cadastro;  
    private String Nome;  
    private int idade;  
  
    public Pessoa(long cadastro, String nome, int idade) {  
        super();  
        this.cadastro = cadastro;  
        Nome = nome;  
        this.idade = idade;  
    }  
  
    public long getCadastro() {  
        return cadastro;  
    }  
  
    public void setCadastro(long cadastro) {  
        this.cadastro = cadastro;  
    }  
  
    public String getNome() {  
        return Nome;  
    }  
  
    public void setNome(String nome) {  
        Nome = nome;  
    }  
  
    public int getIdade() {  
        return idade;  
    }  
  
    public void setIdade(int idade) {  
        this.idade = idade;  
    }  
}
```


Primeiro criamos a superclasse pessoa e damos os atributos cadastro, nome e idade que serão compartilhados por outras classes. O próximo passo é modificar a classe aluno para que a mesma possa herdar os atributos

```
public class Estudante extends Pessoa{  
    private String turma;  
    private boolean matriculaAtiva;  
    ➤ public Estudante(long cadastro, String nome, int idade, String turma) {  
        super(cadastro, nome, idade);  
  
        this.turma = turma;  
        this.matriculaAtiva = true;  
    }  
    ➤ public String getTurma() {  
        return turma;  
    }  
    ➤ public void setTurma(String turma) {  
        this.turma = turma;  
    }  
    ➤ public boolean isMatriculaAtiva() {  
        return matriculaAtiva;  
    }  
    ➤ public void setMatriculaAtiva(boolean matriculaAtiva) {  
        this.matriculaAtiva = matriculaAtiva;  
    }  
}
```

Como podemos perceber o construtor tem um **super** com todos os atributos da super classe e complementamos com os atributos da própria classe estudante. Note que não precisamos reescrever também os métodos assessores, já que estudante já herda da classe pessoa que por sua vez torna-se uma classe genérica a ser herdada por outras classes.

Levantamos então um ponto. Porque instanciar uma classe genérica como Pessoa, tem alguma importância para o nosso sistema?

3.2.1 Classes e métodos abstratos

Para que a classe pessoa não possa ser instanciada usamos a palavra reservada **abstract** tornando assim Pessoa um modelo para as classes que dela herdam. Ao colocarmos o modificador **abstract** em um método e tornando-o abstrato. Estaremos passando para as classes filhas apenas a assinatura de um método que deverá ser implementado pela mesma.

3.2.2 Atributos e métodos estáticos

Um atributo estático é um atributo que pertence a classe e não ao objeto. Ele não será referenciado por **this** e será compartilhado por todas as instâncias da classe. No exemplo vemos que a classe pessoa tem um atributo estático população e no construtor fazemos o incremento dela, ou seja, cada vez que um objeto Estudante ou qualquer outra classe herdeira for instanciado o atributo população vai aumentar em +1.

```
public abstract class Pessoa {  
  
    private long cadastro;  
    private String Nome;  
    private int idade;  
    private static int populacao;  
  
    public Pessoa(long cadastro, String nome, int idade) {  
        super();  
        this.cadastro = cadastro;  
        Nome = nome;  
        this.idade = idade;  
        populacao++;  
    }  
}
```

Criaremos então uma função estática mostraPopulação que vai imprimir o número de objetos instanciados. Por ser um método estático não precisamos instanciar um objeto para usá-lo, precisaremos apenas chamar a classe e depois o método.

```
public void setIdade(int idade) {  
    this.idade = idade;  
}  
  
public static int mostraPopulacao() {  
    return populacao;  
}
```

Por fim, instanciamos três objetos do tipo Estudante e chamamos a função mostraPopulacao(); Que nos retornará o valor três.

```
package cursoJava.orientacaoObjeto;

public class TesteClasses {

    public static void main(String[] args) {

        Estudante e1 = new Estudante(233,"José da Silva",22,"A");
        Estudante e2 = new Estudante(233,"Maria Antonieta",18,"A");
        Estudante e3 = new Estudante(233,"Lucia Matos",21,"B");

        e1.setTurma("B");
        e1.setIdade(19);

        //e1.boletim(e1.media(10, 4));

        System.out.println("Estudantes matriculados: "+Estudante.mostraPopula
    }
}
```

<terminated> TesteClasses [Java App
Estudantes matriculados: 3

3.3. Interfaces e herança múltipla.

Como explicado anteriormente os métodos definem o comportamento do objeto com o mundo externo, o como ele vai responder a partir de suas próprias ações. Interface são estruturas que contém um grupo de métodos com o corpo vazio. Ela apenas especifica que métodos o nosso objeto deve ter. Podemos aplicar um método de um objeto através da palavra-chave implements a classe que recebe uma interface deve então implementar seus métodos e pode, inclusive, implementar mais de uma classe.

```
package cursoJava.orientacaoObjeto;

public interface GerenciadorEstudantes {

    void presenca(boolean presenca);
    void declaracao();
    double media(double nota1, double nota2);
}
```

```
package cursoJava.orientacaoObjeto;

public interface GerenciadorSecretaria {

    void boletim(double nota);
    void declaracao();
    void cancelamento();
}
```

Foram criados duas interfaces uma com os métodos da secretaria e outra de gerenciamento do cadastro de alunos. Note que todos os métodos de uma interface são abstratos e agora nós implementamos ambas em estudante e sobrescrevemos os métodos não implementados.

```
public class Estudante extends Pessoa implements GerenciadorEstudantes, GerenciadorSecretaria{  
  
    private String turma;  
    private boolean matriculaAtiva;  
    private int presenca;  
    private int falta;  
  
    public Estudante(long cadastro, String nome, int idade, String turma) {  
        super(cadastro, nome, idade);  
  
        this.turma = turma;  
        this.matriculaAtiva = true;  
    }  
}
```

```
@Override  
public void cancelamento() {  
    this.matriculaAtiva = false;  
}  
  
@Override  
public void presenca(boolean presenca) {  
    if(presenca)  
        this.presenca++;  
    else  
        this.falta++;  
}
```

3.4. Exercícios propostos

1) Qual a palavra reservada para que uma classe herde os atributos e métodos de outra classe?

- a) implements
- b) extends
- c) abstract
- d) static
- e) this

2) É a palavra reservada que faz com que atributos passem a pertencer a classe e métodos sejam chamados sem precisar de uma instância de objeto.

- a) implements
- b) extends
- c) abstract
- d) static
- e) this

3) É a palavra reservada utilizada para que a classe receba um conjunto de métodos para implementar através de uma interface.

- a) implements
- b) extends
- c) abstract
- d) static
- e) this

4) Essa palavra reservada aponta para o atributo do objeto ao qual ela é referenciada.

- a) implements
- b) extends
- c) abstract
- d) static
- e) this

5) Aplicada a uma classe essa palavra reservada faz com que essa classe não possa mais ser instanciada por objetos, servindo então apenas de modelos para as classes filhas e se aplicada a métodos permite que o mesmo seja declarado apenas como uma assinatura que só será implementado pelas classes filhas.

- a) implements
- b) extends
- c) abstract
- d) static
- e) this

6) Quando uma classe herda os atributos e métodos de outra dentro do construtor devemos chamar os atributos da classe mãe, o fazemos utilizando qual palavra reservada?

- a) this
- b) break
- c) interface
- d) String
- e) boolean

7) Faça a classe produtos com os atributos: cadastro; tipo; quantidade (estática); preço; e os métodos: estoque (quantidade++); preço (quantidade- -); detalhes; Depois crie duas classes filhas eletrônicos os atributos adicionais consumo e garantia e a classe frutas com o atributo adicional validade.

8) Faça quatro objetos de entidades relacionadas ao seu dia a dia. Exemplo: casa.

9) Crie as classes ave, mamífero e réptil e as faça herdar atributos e métodos da classe animal.

10) Crie uma interface com métodos aplicáveis as figuras geométricas (área, comprimento) e implemente nas classes quadrado, triângulo e losango.

4. Arrays, listas e coleções.

4.1. Arrays

..Arrays, ou vetores são variáveis que armazenam mais de um valor. Servem para quando precisamos de vários registros de uma mesma variável como uma sala com trinta alunos. Você não criaria uma variável para cada, apenas faria uma lista com o nome de todos eles. Neste capítulo discorreremos dos arrays desde a sua forma mais simples até chegarmos aos hashmaps e arraylist.

```
package cursoJava.basico;

public class Arrays {

    public static void main(String[] args) {

        char vogais[] = {'a','e','i','o','u'};

        System.out.println("Valor inicial: "+vogais[1]);

        vogais[1] = 'E';

        System.out.println("Valor modificado: "+vogais[1]);

        //percorrendo o array e imprimindo todos os valores.
        for(char vogal : vogais) {

            System.out.println(vogal);

        }

    }

}
```

<terminated> Arrays [Java App
Valor inicial: e
Valor modificado: E
a
E
i
o
u

Podemos ver todo o processo de criação e inicialização dos valores de um vetor de nome “vogais” e cujo cada elemento contido dentro dele é do tipo **char**, o acesso a um determinado índice e a alteração do mesmo, e por fim, um for para imprimir todos os valores dentro dele.

Ao criarmos um vetor podemos fazê-lo especificando o número de posições que ele terá ou deixando implícito ao inicializar seus valores. O índice inicial de um vetor sempre será 0 e o final será o tamanho dele menos um (10 posições, 0 a 9).

4.2. List e ArrayList

A primeira grande diferença do Array para o ArrayList é que é possível modificarmos o tamanho do segundo e temos vários métodos para a sua manipulação e o array uma vez criado é imutável no que tange ao seu tamanho.

```
public static void main(String[] args) {  
  
    ArrayList<Integer> pares = new ArrayList<>();  
  
    pares.add(2);  
  
    pares.add(6);  
  
    pares.add(1, 4);  
  
    pares.add(24);  
  
    pares.add(10);  
  
    pares.add(8);  
  
    for(int numero : pares) {  
        System.out.print(numero+" ");  
    }  
  
    System.out.println();  
}
```

Podemos ver a criação de um arrayList do tipo interger e nome pares. Adicionamos a ele na primeira posição o valor 2 na segunda 6 e logo após adicionamos o número 4 na segunda posição, tornando o 6 o terceiro. Logo abaixo temos o for que irá percorrer o arraylist e imprimir seus valores.

```
pares.sort(null);  
  
pares.remove(4);  
  
System.out.println(pares.indexOf(24));  
  
System.out.println(pares.get(1));
```

logo temos:

- **add()** para adicionar um novo valor na última posição.
- **add(índice, valor)** que adiciona um valor em um índice pré-determinado
- **sort()** vai ordenar o arraylist por ordem alfabética ou ordem numérica.
- **remove(índice)** remove o item correspondente ao índice no arraylist.
- **indexOf(valor)** recebe um valor e procura no arraylist se existe um item correspondente e retorna seu índice.
- **array.get(índice)** retorna o valor de determinado índice.

4.3. Exercícios propostos

1) Qual a função que adiciona um valor na última posição do arrayList?

- a) .sort();
- b) .indexOf(valor);
- c) remove(índice);
- d) .get(índice);
- e) .add();

2) Qual a função que remove um valor de um determinado índice no arrayList?

- a) .sort();
- b) .indexOf(valor);
- c) remove(índice);
- d) .get(índice);
- e) .add();

3) Qual a função que retorna o valor de uma determinada posição do arrayList?

- a) .sort();
- b) .indexOf(valor);
- c) remove(índice);
- d) .get(índice);
- e) .add();

4) Qual a função que organiza o arrayList em ordem alfabética ou em ordem numérica?

- a) .sort();
- b) .indexOf(valor);
- c) remove(índice);
- d) .get(índice);
- e) .add();

5) Crie um arrayList do tipo inteiro e coloque dentro dele os dez primeiros quadrados perfeitos ex: 4, 9, 16, 25, 36...

6) Crie um array que receba o inverso do arrayList de vogais = {'a','e','i','o','u'};

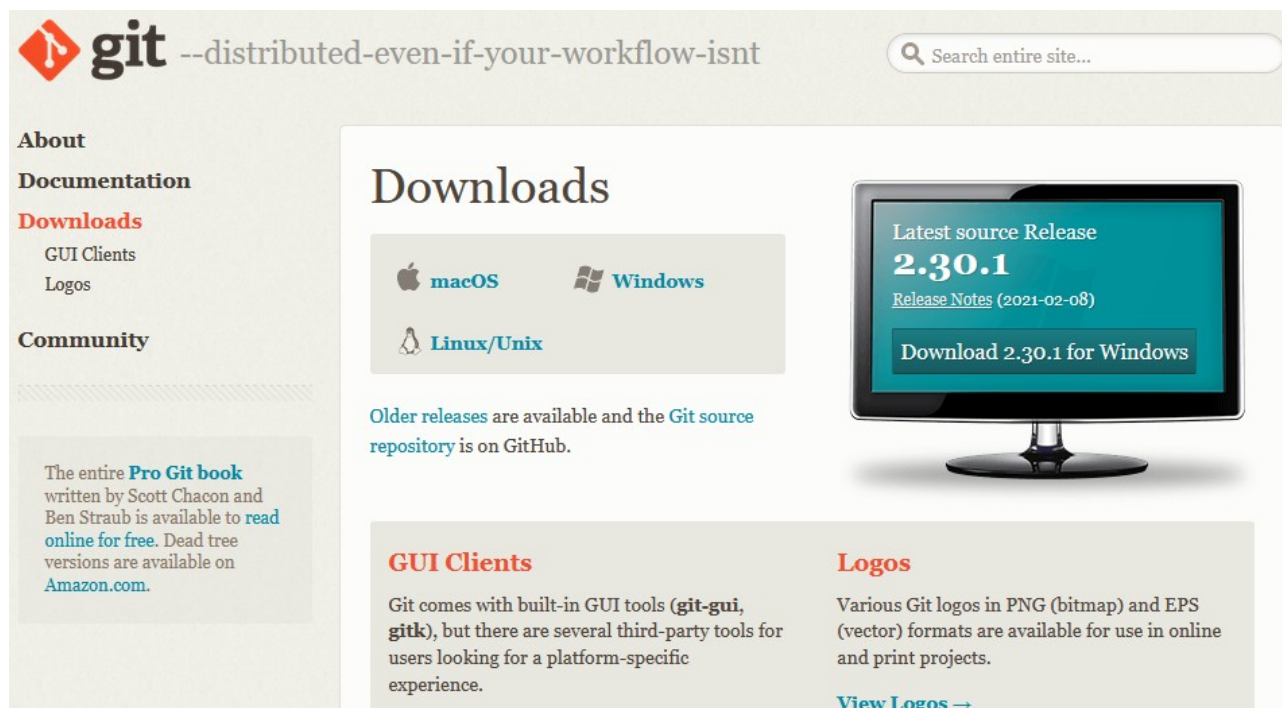
7) Crie uma variável que receba a soma do array feito na quinta questão.

8) Faça cópia para um array de nome par somente os valores pares do array numeros = {21,22,23,24,25,26,27,28,29,30};

9) Crie um vetor de alunos e organize por ordem alfabética.

5. Git & GitHub

Ambas são ferramentas de controle e versionamento de projetos. Muito utilizadas no mundo da programação para gerência de projetos e ferramentas de colaboração. Para instalar o Git no computador você deve ir ao endereço: <https://git-scm.com/downloads> e baixar o git, logo após instalar.



Siga as instruções de instalação. Você irá escolher o diretório onde será instalado e depois de finalizada a instalação basta abrir o prompt de comando e digitar: **git - - version**

5.1 Lista de principais comando Git.

Setar os dados do usuário:

```
git config --global user.name "João da silva"
```

```
git config --global user.email joao@example.com
```

Iniciar um repositório:

```
git init
```

Adicionar arquivos para poder commitar

```
git add <nome do arquivo>
```

```
git add *
```

Commitar as mudanças para o cabeçalho, mas não para o repositório online

```
git commit -m "mensagem que identifica o commit"
```

Adicionar a origem de um repositório on line.

```
git remote add origin <endereço do site>
```

Remover a origem do repositório

```
git remote remove origin
```

Adicionar os arquivos do repositório local para o repositório on line.

```
git push origin master
```

Checar o status dos arquivos

```
git status
```

Criar um novo branch ao repositório

```
git checkout -b <nome do branch>
```

Trocar para um branch já existente

```
git checkout <nome do branch>
```

Listar todos os branches

```
git branch
```

Adicionar o branch ao seu repositório remoto

```
git push origin <nome do branch>
```

Adicionar todos os branches para o seu repositório remoto

```
git push - -all origin
```

Deletar um branch do seu repositório remoto

```
git push origin :<nome do branch>
```

5.2. Exercícios propostos

1) É uma ferramenta de controle de versionamento e gerência de arquivos on line. Que serve também de repositório para os seus projetos.

- a) Git
- b) Java EE
- c) GitHub
- d) ArrayList
- e) JVM

2) Qual o comando git para adicionar o endereço do seu repositório on line no seu repositório local.

- a) git init
- b) git remote add origin "endereço"
- c) git remote remove origin
- d) git push origin master
- e) git status

3) Qual o comando git para iniciar o repositório local?

- a) git init
- b) git remote add origin "endereço"
- c) git remote remove origin
- d) git push origin master
- e) git status

4) Qual o comando que utilizamos para enviar nossos arquivos para o repositório na nuvem?

- a) git init
- b) git remote add origin "endereço"
- c) git remote remove origin
- d) git push origin master
- e) git status

5) Comando git utilizado para checar o status dos seus arquivos?

- a) git init
- b) git remote add origin "endereço"
- c) git remote remove origin
- d) git push origin master
- e) git status

6) Comando para apagar a origem remota do seu repositório localiza

- a) git init
- b) git remote add origin "endereço"
- c) git remote remove origin
- d) git push origin master
- e) git status

7) Comando utilizado para criar um novo branch no repositório.

- a) git add
- b) git branch
- c) git checkout -b <nome do branch>
- d) git push origin :<nome do branch>
- e) git status

8) Comando que mostra todos os branches no repositório.

- a) git add
- b) git branch
- c) git checkout -b <nome do branch>
- d) git push origin :<nome do branch>
- e) git status

9) Deletar o branch no repositório remoto.

- a) git add
- b) git branch
- c) git checkout -b <nome do branch>
- d) git push origin :<nome do branch>
- e) git status

10) Adicionar os arquivos para staging

- a) git add
- b) git branch
- c) git checkout -b <nome do branch>
- d) git push origin :<nome do branch>
- e) git status