4LINUX
FREE SOFTWARE SOLUTIONS

Introdução à linguagem Java

Curso 470

www.4linux.com.br

Sumário

Índice de tabelas

Índice de Figuras

Java Básico

1.1. Objetivos:

- Saber compilar um código Java
- Entender a estrutura sintatica:classes, métodos e variaveis;
- O uso do JavaDoc
- Usar o Eclipse para Desenvolvimento

1.2. Sobre o SVN

Um sistema de controle de versão (VCS - *Version Control System*) é um software capaz de controlar as alterações feitas em arquivos binários ou texto de um projetos. O VCS mantém um histórico de todas as mudanças feitas ao longo do processo de desenvolvimento, é possível identificar *quem*, *quando* e *o que* foi alterado, viabilizando o trabalho em equipe além de servir para gerência de projeto.

A idéia é manter todo o diretório do projeto em um repositório externo, os mais populares são: cvs, mercurial, bazar, git, Visual Source Safe, etc. Deste modo, os desenvolvedores envolvidos neste projeto são capazes de acessar os arquivos deste projeto, fazer suas alterações localmente e submete-lo novamente ao repositório. Alguns processo típicos são:

- import: o desenvolvedor importa um projeto local para o servidor;
- checkout: o desenvolvedor cria uma cópia local do código-fonte que se encontra no servidor;
- comit: as alterações são enviadas para o servidor. Conflitos de alterações podem ocorrer e são resolvidas pelo próprio desenvolvedor.

é um sistema de controle de versão criado para superar seu antecessor, "CVS". Este é um software livre, disponível em: *http://subversion.tigris.org/*, desenvolvido pela empresa CollabNet e atualmente está na versão 1.6. Muita empresas adotam este software para controle de versão, outras preferem softwares comerciais, tais como, ClearCase (da IBM) ou SourceSafe (da Microsoft). Esta preferência está relacionada com a garantia, pois software livres não se responsabilizam por perdas das informações ou erros no software.

Seu funcionamento é baseado em um Servidor Subversion que armazena todo os arquivos e diretórios de um projeto, normalmente arquivos de configurações, códigos fontes, documentos de projeto, bibliotecas. Permite ver o histórico de mudanças - data de modificação, usuário e observação. Não permite edição direta do arquivo, sendo normalmente associado a um URL, podendo ser acessado via HTTP(S), SVN e compartilhado de arquivos em rede, quando utilizando a internet,

acesso normal via HTTPS ou SVN+SSH.

1.3. O uso de SVN no curso

No curso usaremos o SVN para que você possa baixar os exercícios e quando tiver concluído enviar para o repositorio assim o instrutor poderá corrigir, alem de ser uma comprovação de entrega. Em uma equipe de desenvolvimento é muito difícil manter um software sem controle de versão.

1.4. Estrutura Básica do Java

Veremos a estrutura básica do Java com mais detalhes nessa apostila, portanto leia com bastante atenção e anote as dúvidas. O pré-aula fizemos apenas uma introdução ao assunto.

1.5. Classes

O conceito de classe é um dos alicerces da linguagem Java. Isso é consequência do fato de que Java é totalmente orientado a objetos. Uma classe define a forma e a natureza de um objeto. Por isso, o conceito de classe forma a base da programação orientada a objetos em Java.

Qualquer conceito que se queira implementar em um programa Java deve ser encapsulado dentro de uma classe. Pelo fato das classes serem tão fundamentais em Java, esta lição e as próximas tratarão deste assunto com profundidade.

Na verdade, no momento em que você escreveu seu primeiro programa em Java, você já fez uso das classes. Porém, até agora, as classes somente foram utilizadas em sua forma mais elementar. As criadas nos exemplos anteriores basicamente existiam simplesmente para encapsular o método main(), que tem sido usado para demonstrar o básico da sintaxe de Java.

Veremos, agora, que as classes são muito mais poderosas do que aquilo que foi demonstrado anteriormente. Talvez o mais importante a entender sobre as classes é que elas definem um novo tipo de dado. Por exemplo, se criarmos uma classe

chamada “NovoTipo”, podemos a partir daí criar objetos do tipo “NovoTipo”.

Portanto, uma classe é a definição de um objeto, e um objeto é um exemplar de uma classe. Como um objeto é uma instância de uma classe, muitas vezes veremos essas duas palavras, objeto e instância, sendo usadas de forma intercambiável.

Quando definimos uma classe, precisamos declarar com exatidão sua forma e sua natureza. Isso é feito especificando-se os dados que ela contém e as funções (métodos) que operam sobre esses dados. Embora classes muito simples possam conter apenas funções (métodos) ou apenas dados (variáveis), no mundo real, a maior parte das classes contém ambos. Como veremos, o código de uma classe define a interface utilizada para acessar seus dados.

Uma classe é declarada com o uso da palavra-chave `class`. As classes que usamos até este ponto são na verdade exemplos muito limitados da forma completa. Como veremos, as classes podem ser muito mais complexas, e geralmente são.

Não se preocupe muito se alguns termos por agora não fazerem sentindo, ou, estranho. A medida que vamos avançando no curso e praticando as dúvidas teoricas tendem a serem solucionadas através da pratica.

Qual comando para compilar uma classe Java no prompt de comado?

O comando javac é responsável por compilar e o comando java por executar

Quando pensar em uma classe, pense em você, seu carro, seu cachorro. Todos são uma classe, já que cada um possuem caracteristicas(os atributos)

A programação Orientação a Objetos tem como característica imitar o "mundo real" por meio de abstrações. Como primeiro exemplo, iremos utilizar uma classe muito

simples. Criaremos criar uma classe que representa uma "Pessoa" e, coerentemente, será chamada de classe Pessoa. Eis a definição da classe Pessoa:

```
class Pessoa {
        double largura;
        double altura;

} // fim da classe Pessoa
```

Observe que uma "Pessoa real" poderia ter várias outras características, como: idade; peso; etc. Nesse nosso exemplo, limitamos nossa abstração somente às propriedades largura, altura.

Nossa classe Pessoa define duas variáveis de instância: “largura” e “altura”. Variáveis de instância é o nome que damos às variáveis ou dados contidos dentro de uma classe. São elas que definem as propriedades de um objeto da classe Pessoa. Observe que, por enquanto, a classe Pessoa não contém nenhum método.

Como dissemos, uma classe define um novo tipo de dado. Nesse caso, o novo tipo de dado é chamado Pessoa. Usaremos esse nome para declarar objetos do tipo Pessoa.

É importante lembrar que a declaração de uma classe apenas não cria um objeto. Assim, a declaração não faz com que um objeto do tipo Pessoa passe a existir. Para que seja criado, de fato, um objeto do tipo Pessoa, usamos o seguinte comando:

```
Pessoa pessoa1 = new Pessoa();  // cria um objeto do tipo Pessoa chamado pessoa1.
```

Depois que o comando for executado, "pessoa1" será uma instância de Pessoa, passando a ser uma realidade física. Por enquanto, não se preocupe com os detalhes desse comando.

Novamente, cada vez que criamos uma instância de uma classe, estamos criando um objeto que contém sua própria cópia de cada variável de instância definida pela classe. Portanto, cada objeto Pessoa contém cópias próprias das variáveis de instância “largura” e “altura”.

Para acessar essas variáveis, usamos o operador “ponto” ('.'). Ele conecta o nome do objeto ao nome de uma variável de instância. Por exemplo, para atribuir o valor 50 à variável “largura” de “pessoa1”, usamos o seguinte comando:

```
Pessoa1.largura = 50;
```

Eis o que esse comando diz ao compilador: atribua à cópia da variável “largura” que está contida dentro do objeto “pessoa1” o valor 50.

Em geral, utilizamos o operador “ponto” para acessar tanto as variáveis de instância quanto os métodos de um objeto. Eis um programa completo que utiliza a classe Pessoa:

```
// Ilustra o uso da classe Pessoa.
class Pessoa {
        double largura;
        double altura;

} // fim da classe Pessoa
public class PessoaDemo {
        public static void main(String args[]) {
                // Declara um objeto da classe Pessoa, chamado Pessoa1.
                Pessoa Pessoa1 = new Pessoa();
                double volume;
                // Atribui valores às variáveis do objeto Pessoa1.
                Pessoa1.largura = 5;
                Pessoa1.altura = 10;
                Pessoa1.profundidade = 20;
                // Calcula o volume de Pessoa1.
                volume = Pessoa1.largura * Pessoa1.altura * Pessoa1.profundidade;
                // Exibe o volume.
                System.out.println( "Volume de Pessoa1 = " + volume );
        } // fim do método main
} // fim da classe PessoaDemo
```

Quando esse programa for compilado, veremos que dois arquivos “.class” foram criados:

- *PessoaDemo.class*
- *Pessoa.class*

O compilador Java automaticamente coloca cada classe em um arquivo “.class” separado, não importando se originalmente as duas classes, “Pessoa” e “PessoaDemo”,

estão ou não no mesmo arquivo fonte.

Se quiséssemos, poderíamos colocar cada classe em seu próprio arquivo fonte, chamando-os de “Pessoa.java” e “PessoaDemo.java”, respectivamente.

Para rodar esse programa, devemos executar a classe “PessoaDemo.class”, porque é ela que contém o método main(), e isso é feito com o seguinte comando:

```
java PessoaDemo
```

O programa gera a seguinte saída:

```
Volume de Pessoa1 = 1000.0
```

Observação: se estiver usando o Eclipse, não precisa digitar nenhum comando, a própria IDE cuida disso.

1.6. As variaveis

Em Java, toda variável tem um tipo que não pode ser mudado, uma vez que declarado:

tipoDaVariavel nomeDaVariavel;

Por exemplo, é possível ter uma idade que guarda um número inteiro:

int codigo;

Com isso, você declara a variável codigo, que passa a existir a partir daquela linha. Ela é do tipo int, que guarda um número inteiro. A partir de agora, você pode usá-la, primeiramente atribuindo valores.

A linha a seguir é a tradução de: *“a variavel codigo em o valor quinze”.*

codigo = 15

Alem de atribuir podemos imprimir esse valor usando *System.out.println* e usar o nome da variável.

System.out.println(codigo);

Para alterar o valor de uma variavel, basta usarmos o sinal = e definir o novo valor, mas também podemos aproveitar o valor atual e adicionar veja:

```
codigo = codigo + 1
```

Estamos dizendo que a variavel codigo terá o valor atual somado com 1. Sendo agora o valor 16. Para outras operações matematicas basta usarmos os sinais: - / % *.

Em Java temos 8 tipos primitivos que são eles:

TIPO	TAMANHO
boolean	1 bit
byte	2 byte
short	2 bytes
char	2 bytes
int	4 bytes
float	4 bytes
long	8 bytes
double	8 bytes

1.7. Variáveis de instância

Cada objeto tem suas próprias cópias das variáveis de instância. Isso significa que se tivermos dois objetos Caixa, cada um deles tem sua própria cópia das variáveis “largura”, “altura” e “profundidade”.

Alterações nas variáveis de instância de um objeto não têm efeito sobre as variáveis de instância de outro objeto.

O programa abaixo ilustra esse fato. Ele declara dois objetos da classe “Caixa”.

```
// CaixaDemo2.java

// Ilustra a separação entre variáveis de instância.

class Caixa {
```

```
        double largura;
        double altura;
        double profundidade;
} // fim da classe Caixa
public class CaixaDemo2 {
        public static void main(String args[]) {
                // Declara dois objetos da classe Caixa, chamados caixa1 e caixa2.
                Caixa caixa1 = new Caixa();
                Caixa caixa2 = new Caixa();
                double volume1, volume2;
                // Atribui valores às variáveis de instância do objeto caixa1.
                caixa1.largura = 5;
                caixa1.altura = 10;
                caixa1.profundidade = 20;
                // Atribui valores às variáveis de instância do objeto caixa2.
                caixa2.largura = 10;
                caixa2.altura = 15;
                caixa2.profundidade = 25;
                // Calcula o volume de caixa1.
                volume1 = caixa1.largura * caixa1.altura * caixa1.profundidade;
                // Calcula o volume de caixa2.
                volume2 = caixa2.largura * caixa2.altura * caixa2.profundidade;
                // Exibe o volume de caixa1.
```

```
            System.out.println( "Volume de caixa1 = " + volume1 );

            // Exibe o volume de caixa2.

            System.out.println( "Volume de caixa2 = " + volume2 );

        } // fim do método main

} // fim da classe CaixaDemo2
```

Baixe o código-fonte acima neste

A saída de “CaixaDemo2.java” é a seguinte:

Volume de caixa1 = 1000.0

Volume de caixa2 = 3750.0

É possível perceber que os dados definidos para “caixa1” são completamente independentes dos dados definidos para “caixa2”.

1.8. Métodos

Dissemos, anteriormente, que as classes geralmente consistem em duas coisas: variáveis de instância e métodos. O assunto métodos é muito extenso, porque é deles que o Java obtém muito de seu poder e flexibilidade. Por isso, em diferentes partes deste curso, falaremos sobre diversos aspectos dos métodos.

Nesse momento, você precisa entender alguns fundamentos dos métodos, para que possa começar a acrescentar métodos a suas classes.

Esta é a forma geral de um método:

```
tipo nomeDoMetodo( lista-de-parâmetros )
{
  // Corpo do método.
}
```

“tipo” especifica o tipo de dados retornado pelo método. Pode ser qualquer tipo válido, inclusive os tipos de classes criadas pelo programador. Se um método não retorna nenhum valor, seu valor retornado deve ser

declarado como sendo “void”.

O nome do método, o qual é especificado por “nomeDoMetodo”, pode ser qualquer identificador legal, a não ser aqueles já usados por outros itens dentro do escopo atual.

A “lista-de-parâmetros” é uma sequência de pares compostos de um tipo e um identificador, separados por vírgulas. Os parâmetros são, essencialmente, variáveis que recebem o valor dos argumentos passados para o método quando ele é chamado. Se o método não tem parâmetros, os parênteses deverão estar vazios.

Métodos que retornam um tipo que não seja “void” retornam um valor para a rotina que os chama, usando a seguinte forma do comando “return”:

```
return valor;
```

onde “valor” é o valor retornado.

Raramente criamos uma classe como “Caixa”, que contém somente dados, embora isso seja perfeitamente legal em Java. Na prática, geralmente usamos métodos para acessar as variáveis de instância definidas pela classe.

São os métodos que definem a interface da maioria das classes. Isso permite que o implementador da classe oculte o layout interno das estruturas de dados da classe, por trás das abstrações de métodos. Isso é muito mais seguro e elegante.

Além de definir métodos que proporcionam acesso aos dados, também podemos definir métodos que são usados internamente pela própria classe. Vamos começar acrescentando um método à classe “Caixa”.

Observe novamente o programa “CaixaDemo.java”. Dentro da filosofia da orientação a objetos, é muito mais lógico que o cálculo do volume da caixa seja feito pela própria classe “Caixa”, e não pela classe “CaixaDemo”. Como o volume de uma caixa depende de suas dimensões, faz sentido deixar que a classe “Caixa” o calcule.

Vamos fazer isso, acrescentando um método à classe “Caixa”, conforme mostra o exemplo abaixo:

```
// CaixaDemo3.java

// Ilustra o uso de um método simples.
```

```
class Caixa {

    double largura;

    double altura;

    double profundidade;

    // Calcula e exibe o volume da caixa.

    void volume() {

        System.out.println( "Volume = " + (largura * altura *
profundidade) );

    } // fim do método volume()

} // fim da classe Caixa

public class CaixaDemo3 {

    public static void main(String args[]) {

        // Declara dois objetos da classe Caixa, chamados caixa1 e caixa2.

        Caixa caixa1 = new Caixa();

        Caixa caixa2 = new Caixa();

        // Atribui valores às variáveis do objeto caixa1.

        caixa1.largura = 5;

        caixa1.altura = 10;

        caixa1.profundidade = 20;

        // Atribui outros valores às variáveis do objeto caixa2.

        caixa2.largura = 8;

        caixa2.altura = 12;

        caixa2.profundidade = 16;
```

```
            // Calcula e exibe o volume de caixa1, usando o método volume().
            caixa1.volume();
            // Calcula e exibe o volume de caixa2, usando o método volume().
            caixa2.volume();
        } // fim do método main()
    } // fim da classe CaixaDemo3
```

Este programa gera a seguinte saída:

```
Volume = 1000.0
Volume = 1536.0
```

Observe as seguintes linhas:

```
caixa1.volume();
caixa2.volume();
```

A linha

```
caixa1.volume();
```

invoca o método “volume()” para o objeto “caixa1”. Isto é, ela chama “volume()” em relação ao objeto “caixa1”, usando o nome do objeto seguido pelo operador “ponto”. Portanto, a chamada caixa1.volume() exibe o volume da caixa definida por “caixa1”. Da mesma forma, a chamada

```
caixa2.volume();
```

exibe o volume da caixa definida por “caixa2”.

Cada vez que “volume()” é invocado, ele exibe o volume da caixa especificada. Vamos estudar com mais detalhes o que acontece quando um método é chamado.

Quando caixa1.volume() é executado, o sistema runtime de Java transfere o controle para o código definido dentro de “volume()”. Depois que os comandos que estão dentro de “volume()” são executados, o controle volta para a rotina chamadora,

e a execução é retomada na linha de código que se segue à chamada.

Podemos dizer que o método é a forma usada por Java para implementar sub-rotinas.

Há uma coisa muito importante a se observar dentro do método “volume()”: as variáveis de instância “largura”, “altura” e “profundidade” são referenciadas diretamente, sem necessidade de precedê-las com o nome do objeto nem o operador “ponto”. Quando um método utiliza uma variável de instância que é definida por sua classe, ele faz isso diretamente, sem referência explícita a um objeto e sem usar o operador “ponto”.

Não é difícil entender esse fato. Um método é sempre invocado em relação a um objeto de uma determinada classe, e sempre que isso acontece o objeto passa a ser reconhecido. Assim, dentro de um método, não há necessidade de especificar o objeto uma segunda vez. Isso significa que as variáveis “largura”, “altura” e “profundidade” que aparecem dentro de “volume()” referem-se implicitamente às cópias dessas variáveis encontradas dentro do objeto que invoca “volume()”.

Resumindo:

- Quando uma variável de instância é acessada por um código que não faz parte da classe na qual essa variável de instância é definida, isso precisa ser feito através de um objeto, usando o operador “ponto”.

- Quando uma variável de instância é acessada por um código que faz parte da mesma classe que a variável de instância, essa variável pode ser referenciada diretamente.

O mesmo se aplica aos métodos.

1.9. Retorno de um método

Vimos anteriormente que o método “volume()” calcula o volume de uma caixa no seu devido lugar, que é dentro da classe Caixa.

Porém, isso pode ser implementado de uma forma melhor. Por exemplo, se outra parte do programa precisasse saber o volume de uma caixa, e não exibir seu

valor? Uma forma melhor de implementar “volume()” é fazer com que esse método calcule o volume de uma caixa e retorne o resultado para a rotina que o chama.

É isso que ilustra o exemplo abaixo:

```
// CaixaDemo4.java
// Ilustra uma melhor implementação de um método simples.
class Caixa {
    double largura;
    double altura;
    double profundidade;
    // Calcula e retorna o volume.
    double volume() {
        return largura * altura * profundidade;
    } // fim do método volume()
} // fim da classe Caixa
public class CaixaDemo4 {
    public static void main(String args[]) {
        // Declara dois objetos da classe Caixa, chamados caixa1 e caixa2.
        Caixa caixa1 = new Caixa();
        Caixa caixa2 = new Caixa();
        // Uma variável para conter o valor do volume.
        double volumeVar;
        // Atribui valores às variáveis do objeto caixa1.
        caixa1.largura = 5;
```

```
                caixa1.altura = 10;

                caixa1.profundidade = 20;

                // Atribui outros valores às variáveis do objeto caixa2.

                caixa2.largura = 8;

                caixa2.altura = 12;

                caixa2.profundidade = 16;

                // Calcula e exibe o volume de caixa1, usando o método volume().

                volumeVar = caixa1.volume();

                System.out.println( "Volume de caixa1 = " + volumeVar );

                // Calcula e exibe o volume de caixa2, usando o método volume().

                volumeVar = caixa2.volume();

                System.out.println( "Volume de caixa2 = " + volumeVar);

        } // fim do método main()

} // fim da classe CaixaDemo4
```

Nas linhas

```
volumeVar = caixa1.volume();

volumeVar = caixa2.volume();
```

quando “volume()” é chamado, ele é colocado no lado direito de um comando de atribuição. Do lado esquerdo fica a variável, neste caso “volumeVar”, que receberá o valor retornado por “volume()”. Por isso, depois da linha

```
volumeVar = caixa1.volume();
```

o valor 1000.0, correspondente ao volume de “caixa1”, é armazenado em “volumeVar”. Da mesma forma, depois da linha

volumeVar = caixa2.volume();

"volumeVar" contém o valor 1536.0, correspondente ao volume de "caixa2". Ainda com relação aos valores retornados por um método, dois aspectos importantes devem ser lembrados:

- O tipo do dado retornado por um método deve ser compatível com o tipo retornado especificado para o método. Por exemplo, se o tipo especificado para um método for "boolean", ele não pode retornar um "int".

- A variável que recebe o valor retornado por um método, como "volumeVar" no exemplo acima, também deve ser compatível com o tipo retornado especificado para o método.

Finalmente, observe que "CaixaDemo4.java" poderia ser reescrito de forma um pouco mais eficiente. Na verdade, a variável "volumeVar" é desnecessária. A chamada a "volume()" poderia ter sido usada dentro da própria chamada a "println()", conforme mostrado abaixo:

System.out.println("Volume de caixa1 = " + caixa1.volume());

Dessa forma, quando "println()" for executado, "caixa1.volume()" será chamado automaticamente e seu valor será passado para "println()".

1.10. Métodos com parâmetros

Muitos métodos utilizam parâmetros para realizar suas tarefas. Os parâmetros permitem que um método seja mais genérico, ou seja, um método com parâmetros pode operar sobre diferentes valores de dados e ser usado em diversas situações.

Por exemplo, observe o método abaixo que retorna o quadrado de 2:

```
int quadrado() {

    return 2 * 2;

} // fim do método quadrado()
```

Embora esse método retorne de fato o valor de 2 ao quadrado, seu uso é muito limitado. Contudo, se modificarmos o método de maneira que ele receba um

parâmetro, conforme mostrado a seguir, é possível fazer com que “quadrado()” seja muito mais útil.

```
int quadrado( int i ) {

    return i * i;

} // fim de método quadrado()
```

Agora, “quadrado()” retorna o quadrado de qualquer valor para o qual ele seja chamado. Isto é, “quadrado()” é agora um método de uso geral, capaz de computar o quadrado de qualquer valor inteiro, e não apenas de 2, conforme mostram as linhas abaixo.

```
int x, y;

x = quadrado(7); // x é igual a 49.

x = quadrado(9); // x é igual a 81.

y = 10;

x = quadrado(y); // x é igual a 100.
```

Na primeira chamada a “quadrado()”, o valor 7 é passado para o parâmetro “i”. Na segunda chamada, “i” recebe o valor 9. A terceira chamada passa o valor de “y”, que no caso é 10.

Como mostram esses exemplos, “quadrado()” é capaz de retornar o quadrado de qualquer valor que lhe é passado.

É importante esclarecer o significado dos termos parâmetro e argumento.

Um parâmetro é uma variável definida por um método, e que recebe um valor quando o método é chamado. Por exemplo, em “quadrado()”, “i” é um parâmetro.

Um argumento é um valor passado para um método quando ele é invocado. Por exemplo, “quadrado(10)” passa 10 como argumento. Dentro de “quadrado()”, o parâmetro “i” recebe esse valor.

Vamos aplicar esse novo conceito de métodos parametrizados à classe “Caixa”. Nos exemplos anteriores, as dimensões de cada caixa precisavam ser

definidas separadamente com o uso de uma sequência de comandos:

```
caixa1.largura = 5;

caixa1.altura = 10;

caixa1.profundidade = 20;
```

Embora esse código funcione, ele é trabalhoso de se escrever e sujeito a erros. Por exemplo, o programador pode facilmente esquecer de definir uma dimensão. Além disso, em programas Java bem projetados, as variáveis de instância devem ser acessadas somente através de métodos definidos em sua classe. Assim, digamos, em uma nova versão do programa, podemos alterar o comportamento de um método. Porém, não poderíamos alterar o comportamento de uma variável de instância exposta, sob pena de comprometer o funcionamento de todo o programa.

Portanto, um enfoque melhor para definir as dimensões de uma caixa é criar um método que recebe a dimensão da caixa em seus parâmetros e ajusta cada variável de instância apropriadamente. Esse conceito é implementado no seguinte programa:

```
// CaixaDemo5.java

// Ilustra o uso de um método parametrizado.

class Caixa {

    double largura;

    double altura;

    double profundidade;

    // Calcula e retorna o volume.

    double volume() {

        return largura * altura * profundidade;

    } // fim do método volume()

    // Define as dimensões da caixa.
```

```
        void defineDim( double larg, double alt, double prof ) {
            largura = larg;
            altura = alt;
            profundidade = prof;
        } // fim do método defineDim()
    } // fim da classe Caixa
    public class CaixaDemo5 {
        public static void main(String args[]) {
            // Declara dois objetos da classe Caixa, chamados caixa1 e caixa2.
            Caixa caixa1 = new Caixa();
            Caixa caixa2 = new Caixa();
            // Uma variável para conter o valor do volume.
            double volumeVar;
            // Inicializa cada uma das caixas.
            caixa1.defineDim(5, 10, 20);
            caixa2.defineDim(8, 12, 16);
            // Calcula e exibe o volume de caixa1, usando o método volume().
            volumeVar = caixa1.volume();
            System.out.println( "Volume de caixa1 = " + volumeVar );
            // Calcula e exibe o volume de caixa2, usando o método volume().
            volumeVar = caixa2.volume();
            System.out.println( "Volume de caixa2 = " + volumeVar );
        } // fim do método main()
```

```
} // fim da classe CaixaDemo5
```

Nas linhas

```
caixa1.defineDim(5, 10, 20);

caixa2.defineDim(8, 12, 16);
```

o método “defineDim()” é usado para definir as dimensões de cada caixa. Assim, quando a linha

```
caixa1.defineDim(5, 10, 20);
```

é executada:

- o valor 5 é copiado para o parâmetro “larg”
- o valor 10 é copiado para o parâmetro “alt”
- o valor 20 é copiado para o parâmetro “prof”

Dentro do método “defineDim()”, os valores de “larg”, “alt” e “prof” são atribuídos a “largura”, “altura” e “profundidade”, respectivamente.

1.11. Estrutura Sintática: if/else/while/for/switch

Veremos neste tópico o uso das instruções de controle no Java.

A seguir temos as expressões válidas com IF

- Só podem aceitar valores booleanos
- Em java 0 = = false e 1 = = true *nao é valido.*

```
int trueInt =1;

int falseInt =0;

if(truInt = = true){} // invalido

if(trueInt= = 1){} // valido
```

- A única variável que pode ser atribuída na instrução if é booleana.

```
boolean resultado  = true

if(resultado = false){} // valido

int idade

//invalido

if(idade = 3){}
```

1.12. Switch

- É a maneira de simular varias instruções if
- O **break** é opcional porem causa mudança no resultado na remoção dele.
- **char, short,byte,int ou enum** sao permitido no teste SWITCH
- não é possível compilar se o teste for: **long, float e double.**
- O valor do **case** deve ser constante, ou seja, não pode mudar

```
final int a=1; //valido

final int b;//invalido

b = 2;
```

- Se uma variável for menor que o tipo int é convertido desde, que siga a regra do tipo

```
byte g  =2;

switch(g){

case 23://valido

case 128: // invalido muito grande para um byte.
```

1.13. Break

O importante é que o switch funciona de cima para baixo, e quando um case verdadeiro é encontrado tudo abaixo será executado, isso quando não temos o break para dar um stop no processo.

```
class sw1{

    public static void main(String args[]){

        int x = 3;

        switch(x){

            case 1:

            System.out.println("x is equal to 1");

            break;

            case 2:

            System.out.println("x is equal to 2");

            break;

            case 3:

            System.out.println("x is equal to 3");

            break;

            default:

            System.out.println("valor não encontrado");

        }}}
```

- quando o programa encontra a palavra-chave break ele sai do Switch.
- O processo sem o **break** é chamado de “passagem completa”.

- O uso do break é quando você quer que apenas aquela instrução que for satisfatória seja executada.
- Após a saída do switch o que estiver fora será executada normalmente

1.14. Default

- é se nenhuma instrução case for satisfatória
- A instrução default pode aparecer em qualquer lugar e segue com a mesma regra do break ou sem o break.
- O **default** funciona com passagem completa por isso é recomendável o uso no final do bloco do swtich.

Alguns exemplos:

```
public class sw2{
    public static void main(String args[]){
        String s = "xyz";
        switch(s.length()){
            case 1:
            System.out.println("tamanho 1");
            break;
            case 2: System.out.println("tamanho 2");
            break;
            default:
            System.out.println("sem tamanho");
        }}}
```

No exemplo acima o switch olha o tamanho do objeto string que possui 3 caracteres e compara com os cases, o método que retorna o tamanho de uma *String* é o length() que já vem como padrão na API do Java.

1.15. For

A instrução *for* é usada para percorrer conjunto de dados, que pode ser um array, uma lista, etc. A partir do Java 5, o *for* ganhou um novo apelido : “for básico” o motivo foi que nasceu um novo tipo de for, o chamado de *for aprimorado*, tem o mesmo objetivo, porém melhora a legibilidade quando precisamos percorrer todo o conjunto sem se preocupar em saber o tamanho dele. Veremos mais na frente o *for aprimorado.* Algumas caracteristicas:

- Mais flexível que o for aprimorado;
- é possível colocar mais de uma variável;
- Só pode existir uma expressão TESTE;

```
for(int x=0;(x>5),(y<2);x++){}//invalido
```

Outro exemplo:

```
for(int x=0; x < 3;x++){

System.out.println("in loop");

return true;

}

return true;

}
```

1.16. For aprimorado

- Simplifica a tarefa de fazer um loop através de um array ou conjunto.

- Possui apenas dois componentes:

declaração:variável de um tipo compatível

expressão: avalia o loop. Pode ser: *primitivo, objeto, ou array bidimensional.*

- for(Declaração : expressão){} - for(String s : s){}

Exemplo:

```
class ForEach{
      public static void main(String args[]){
            int[]array={1,2,3};
            for(int a: array)
                  System.out.println(a);

      }}
```

1.17. While

A instrução *while* serve para testar uma condição enquanto ela for verdadeira, a diferença é que sempre que for verdadeira, o que está no bloco do while é executado. Muito cuidado para não criar loops infinitos aqui.

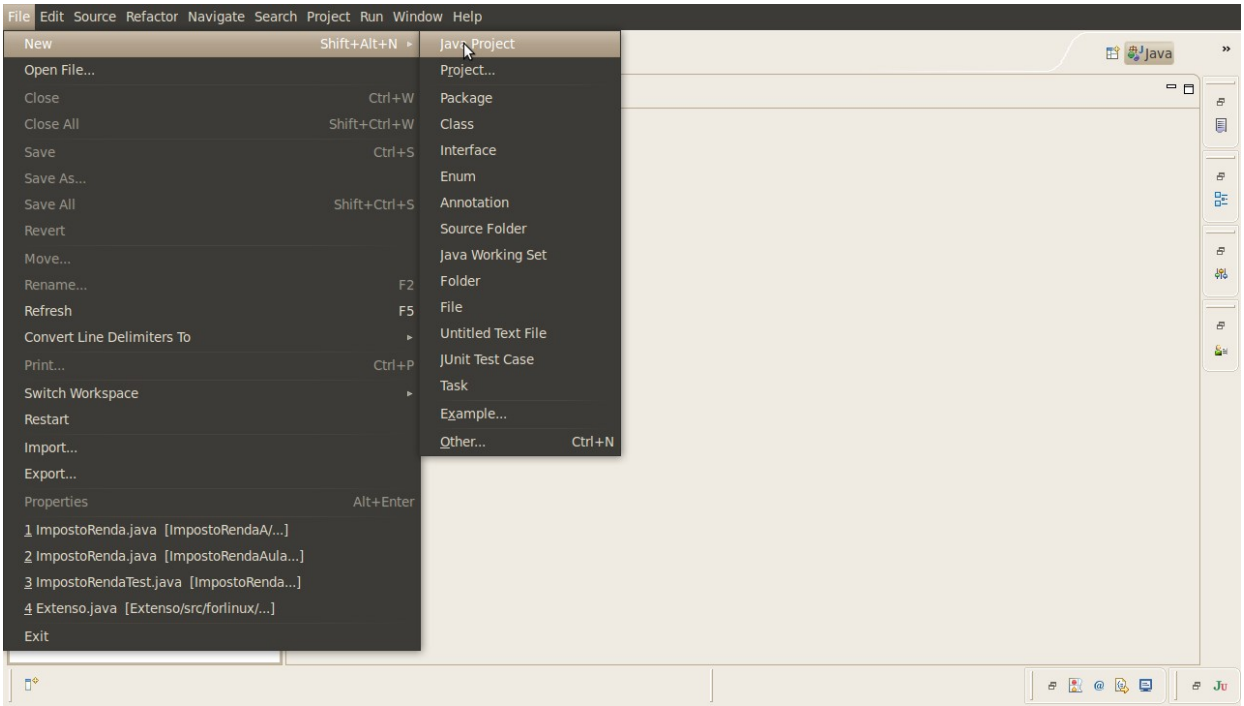
BigDecimal x Float

1.18. Criando um Projeto no Eclipse

Neste tópico veremos como executar um projeto no Eclipse. Levaremos em conta que você tem o Eclipse e o Java instalado na sua máquina, conforme ensinamos na apostila de pré-aula.

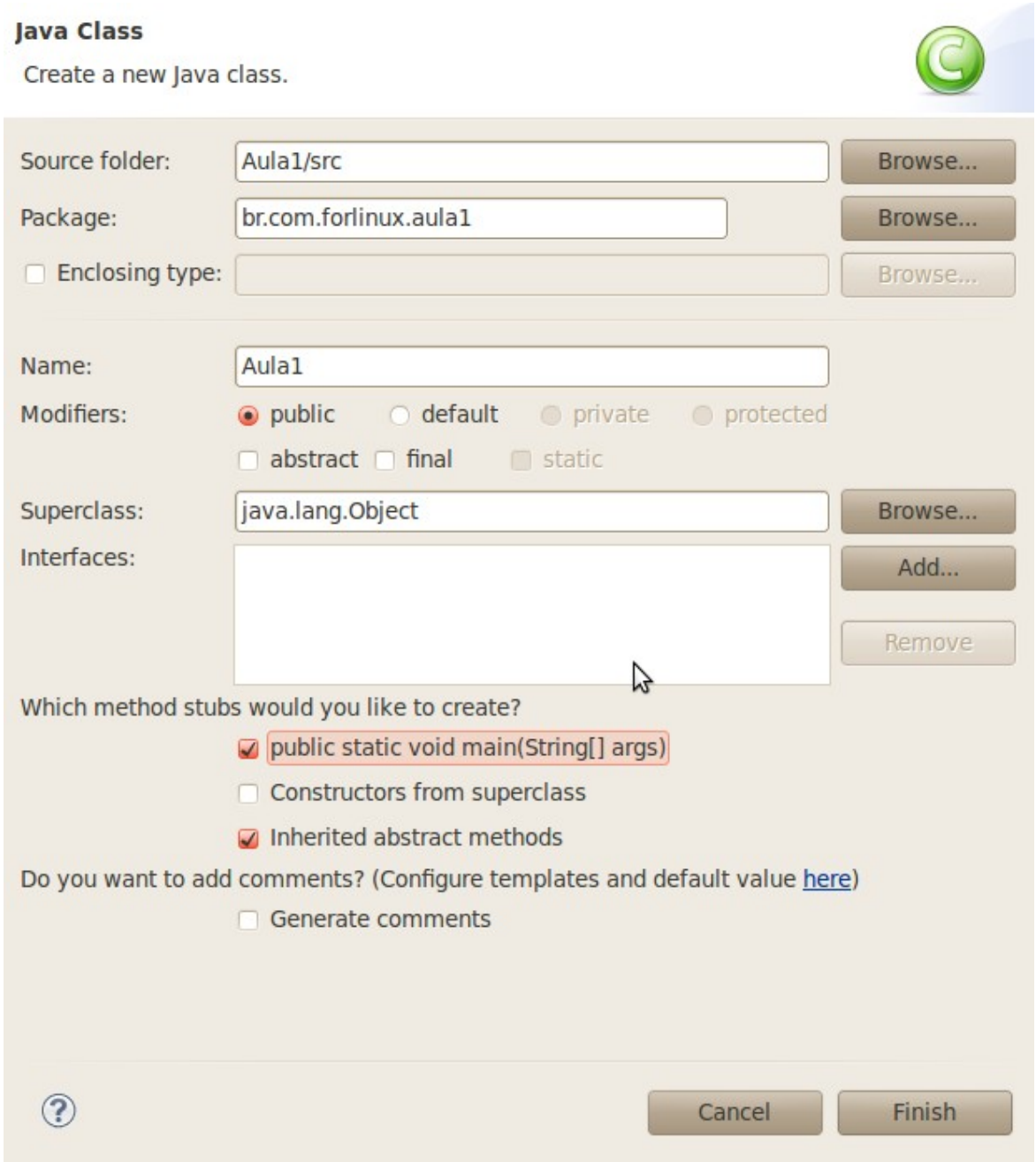
Os nossos printscreen serão baseados em uma distribuição Linux, então é comum ter janelas diferentes, mas nada que afete o seu aprendizado.

1. Abra o Eclipse
2. E carregue seu workspace
3. Agora Crie um Java Project



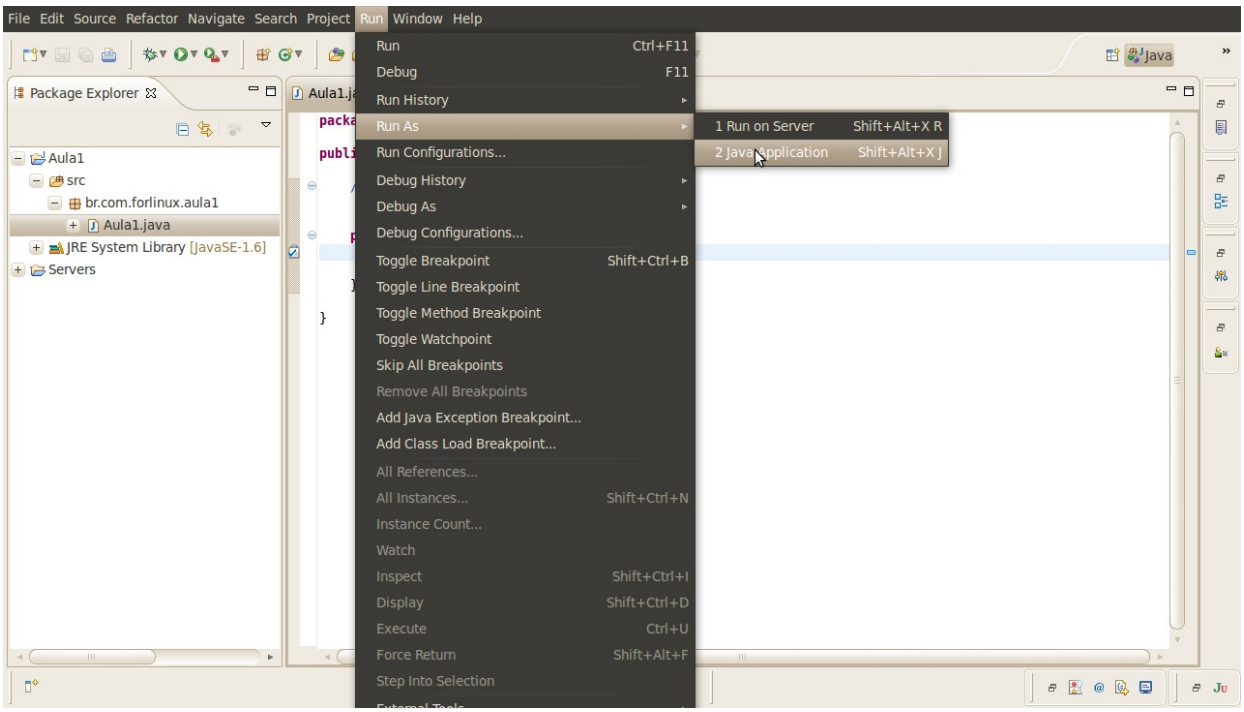
4. Chamaremos o nosso de Aula1
5. Crie um package chamado *br.com.forlinux.aula1*

6. Crie uma classe Java que tenha o método main



7. Crie uma linha que imprima “Hello World 4Linux”

8. Execute a classe



1.19. Termos Técnicos

- construtor padrão (default) é parecido com método sem parâmetros que define como uma classe deve ser instanciada.

- métodos de classe: são também conhecidos como métodos estáticos, ou seja, métodos que são invocados a partir de uma classe. Esse método é marcado com a palavra-chave "static". Temos como exemplo de assinatura de chamada de método estático: Caixa.obtemQuantCaixas().

métodos de instância: são métodos invocados a partir de uma classe. Por exemplo, c.obtemVolume(). Esse método obtém o volume específico de uma instância apontada pela variável c.

New é a palavra-chave que instancia uma classe e sempre precede um método construtor. Ex.: Caixa c = new Caixa();

static é a palavra-chave que indica

variáveis de classe são variáveis de um contexto de classe. Por exemplo: static

int quant_caixas = 0;;

variáveis de instância são variáveis de um contexto de instância. Por exemplo: double largura;

1.20. Exercício

Agora faça o exercício da aula que encontra-se no Redmine. Verifique o ticket informado pelo instrutor. Ao termino do exercício atualize o ticket. Qualquer dúvida, leia o documento de Ambientação Java.