1Z0-808 - Notas de Estudo

Ricardo Alvarenga

2024

Como Tirar a Certificação Entre 3 e 6 Meses de Estudo

Referência: `https://www.youtube.com/watch?v=t-7GVhoM0bY&t=889s&ab_channel=Jos%C3%A9Castilho-DesenvolvedorDisputado`

Por Quê Tirar a Certificação Java

- Poucos profissionais são certificados;
- Gera reconhecimento;
- São duas certificações, a primeira **Java SE Programmer I exam (1Z0-808)**. A segunda querer a primeira e possui várias áreas distintas;

Método Sugerido

- Aprender a linguagem, não começar por livro preparatório. Usar livros básicos primeiro;
- Estudar com data definida;
- "Comer"um livro de certificação;
- Usar várias metodologias. Alguns assuntos são assimilados melhor com leitura, outros com vídeos, outros *codando* e vendo funcionar;
- Fazer muitos simulados. Simular provas até conseguir notas altas em várias provas simuladas;
- Aprender com os erros e acertos dos simulados;
- Repetir o ciclo de fazer simulados e analisar erros e acertos, várias vezes;

Sumário

1 Java Basics

1.1 Define the scope of variables

O escopo é o que determina em que pontos do código uma variável pode ser usada.

1.1.1 Variáveis locais

Chamamos de locais as variáveis declaradas dentro de métodos ou construtores. Antes de continuar, vamos estabelecer uma regra básica: o ciclo de vida de uma variável local vai do ponto onde ela foi declarada até o fim do daquele bloco.

Quando uma variável sai do escopo ela não pode mais ser usada e seu valor é perdido. A qualquer momento o JVM pode realocar a memória ocupada por ela.

Mas o que é um bloco? Podemos entender como bloco um trecho de código entre chaves . Pode ser um método, um construtor, o corpo de um ***if***, de um ***for*** etc.:

```
1 public void m1() { // inicio do bloco do metodo
2     int x = 10; // variavel local do metodo
3 
4     if (x >= 10) { // inicio do bloco do if
5         int y = 50; // variavel local do if
6         System.out.print(y);
7     } // fim do bloco do if
8 } // fim do bloco do metodo
```

Analisando esse código, temos uma variável ***x***, que é declarada no começo do método. Ela pode ser utilizada durante todo o corpo do método. Dentro do ***if***, declaramos a variável ***y***. ***y*** só pode ser utilizada dentro do corpo do ***if***, delimitado pelas chaves. Se tentarmos usar ***y*** fora do corpo do ***if***, teremos um erro de compilação, pois a variável saiu do seu escopo.

Tome cuidado especial com loops ***for***. As variáveis declaradas na área de inicialização do loop só podem ser usadas no corpo do loop:

```
1 for (int i = 0, j = 0; i < 10; i++)
2 j++;
3 
4 System.out.println(j); // erro, ja nao esta mais no escopo
```

Parâmetros de métodos também podem ser considerados variáveis locais ao método, ou seja, só podem ser usados dentro do método onde foram declarados:

```
1  class Teste {
2 
3      public void m1(String bla) {
4          System.out.print(bla);
5      }
6 
7      public void m2() {
8          // erro de compilacao pois bla nao existe neste
9          // escopo
10         System.out.println(bla);
11     }
12 }
```

Os parâmetros podem ser primitivos ou objetos. Um método pode ter até 255 parâmetros. Essas variáveis fazem parte do escopo do bloco do método inteiro e são definidas na declaração do método.

```
float findMilesPerHour(float milesTraveled, float hoursTraveled){
    //o metodo contem dois parametros
    return milesTraveled / hoursTraveled;
}
```

1.1.2 Variáveis de instância

Variáveis de instância ou variáveis de objeto são os atributos dos objetos. São declaradas dentro da classe, mas fora de qualquer método ou construtor. Fazem parte do escopo da classe. Podem ser acessadas por qualquer membro da classe e ficam em escopo enquanto o objeto existir:

```
class Pessoa {
    // variavel de instancia ou variavel de objeto
    String nome;

    public void setNome(String n) {
        // acessando a variavel de instancia no metodo
        this.nome = n;
    }
}
```

1.1.3 Variáveis estáticas (class variables)

Podemos declarar variáveis que são compartilhadas por todas as instâncias de uma classe usando a palavra chave ***static***. Essas variáveis estão no escopo da classe, e lá ficarão enquanto a classe estiver carregada na memória (enquanto o programa estiver rodando, na grande maioria dos casos).

```
class Pessoa {
    static int id = 1;
}

class Teste {
    public static void main(String[] args) {
        Pessoa p = new Pessoa();
        System.out.println(p.id); // acessando pelo objeto. id = 1
        System.out.println(Pessoa.id); // acessando direto pela classe.
                                       // id = 1
        p.id ++; // soma 1
        System.out.println(Pessoa.id); // id = 2
        p.id ++;
        Pessoa pp = new Pessoa();
        p.id ++;
        System.out.println(pp.id); // id = 4
        System.out.println(Pessoa.id); // id = 4
    }
}
```

No caso de variáveis ***static***, não precisamos ter uma referência para usá-las e podemos acessá-las diretamente a partir da classe, desde que respeitando as regras de visibilidade da variável.

1.1.4 Variáveis com o mesmo nome

Logicamente, não é possível declarar duas variáveis no mesmo escopo com o mesmo nome:

```
public void bla() {
    int a = 0;
    int a = 10;// erro
}
```

Mas, eventualmente, podemos ter variáveis em escopos diferentes que podem ser declaradas com o mesmo nome. Em casos em que possa haver ambiguidade na hora de declará-las, o próprio compilador irá emitir um erro evitando a confusão.

Por exemplo, não podemos declarar variáveis de classe e de instância com o mesmo nome:

```
class Bla {
    static int a;
    int a; // erro de compilacao,
}
...
System.out.println(new Bla().a); // qual variavel estamos acessando?
```

Também não podemos declarar variáveis locais com o mesmo nome de parâmetros:

```
public void metodo(String par) {
    int par = 0; // erro de compilacao

    System.out.println(par); // qual?
}
```

Apesar de parecer estranho, é permitido declarar variáveis locais ou parâmetros com o mesmo nome de variáveis de instância ou de classe. Essa técnica é chamada de *shadowing*. Nesses casos, é possível resolver a ambiguidade: para variáveis de classe, podemos referenciar pela própria classe; para variáveis de instância, usamos a palavra chave ***this***:

```
class Pessoa {

    static int x = 0;
    int y = 0;

    public static void setX(int x) {
        // Usando a referencia da classe
        Pessoa.x = x;
    }

    public void setY(int y) {
        // usando o this
        this.y = y;
    }
}
```

Quando não usamos o ***this*** ou o nome da classe para usar a variável, o compilador sempre utilizará a variável de menor escopo:

```
class X {
    int a = 10;

    public void metodo() {
        int a = 20; // shadowing
        System.out.println(a); // imprime 20
    }
}
```

1.1.5 Difference Between *Life* and *Scope* For Local Variables

Life: A local variable is *alive* as long as its Stack frame is on the Stack. In other words, *until the method completes.*

Scope: A local variable is in *scope* only within the method in which the variable was declared. When its own method calls another, the variable is alive, but not in scope until its method resumes. **You can use a variable only when it is in scope**.

Cenário e Solução	
Variável mais adequada para um contador em um laço?	Variável local
Variável deve ser usada para armazenar informações sobre um objeto?	Variável de instância
Que variável deve ser usada para a passagem de informações para um método?	Parâmetro de método

1.2 Define the structure of a Java class

Nesta seção, iremos entender a estrutura de um arquivo java, onde inserir as declarações de pacotes e imports e como declarar classes e interfaces.

Para entender a estrutura de uma classe, vamos ver o arquivo ***Pessoa.java***:

```
// Declaracao de pacote
package br.com.alvarenga.certificacao;

// imports
import java.util.Date;

// Declaracao da classe
class Pessoa {
    // conteudo da classe
}
```

1.2.1 Pacotes

Pacotes servem para separar e organizar as diversas classes que temos em nossos sistemas. Todas as classes pertencem a um pacote, sendo que, caso o pacote não seja explicitamente declarado, a classe fará parte do que chamamos de pacote padrão, ou ***default package***. Todas as classes no ***default package*** se enxergam e podem ser utilizadas entre si. Classes no pacote ***default*** não podem ser importadas para uso em outros pacotes:

```java
// Uma classe no pacote padrao
class Pessoa {
    //...
}
```

Para definir qual o pacote a que a classe pertence, usamos a palavra-chave ***package***, seguida do nome do pacote. Só pode existir um único ***package*** definido por arquivo, e ele deve ser a primeira instrução do arquivo. Após a definição do ***package***, devemos finalizar a instrução com um ***;***. Podem existir comentários antes da definição de um pacote:

```java
// declaracao do pacote
package br.com.alvarenga.certificacao;

class Pessoa {
    //...
}
```

Aproveitando que tocamos no assunto, o ***package*** deve ser a primeira instrução de código que temos declarada em nosso arquivo. Comentários não são considerados parte do código, portanto, podem existir em qualquer lugar do arquivo java sem restrições.

Para inserir comentário em nosso código, temos as seguintes formas:

```java
// comentario de linha

/*
comentario de
multiplas linhas
*/
class /* comentario no meio da linha */ Pessoa {

    /**
    *  JavaDoc, repare que a primeira linha do comentario tem
    *  2 asteriscos
    */
    public void metodo() {
    }
}
```

1.2.2 Classe

Uma classe é a forma no Java onde definimos os atributos e comportamentos de um objeto. A declaração de uma classe pode ser bem simples, apenas a palavra ***class*** seguida do nome e de :

```java
class Pessoa {}
```

Existem outros modificadores que podem ser usados na definição de uma classe, mas veremos essas outras opções mais à frente, onde discutiremos esses modificadores com mais detalhes.

Vale lembrar que java é case *sensitive* e ***Class*** é o nome de uma classe e não podemos usá-lo para definir uma nova classe.

Dentro de uma classe, podemos ter variáveis, métodos e construtores. Essas estruturas são chamadas de **membros da classe**.:

```java
class Pessoa {

    String nome;
    String sobrenome;

    Pessoa(String nome, String sobrenome) {
        this.nome = nome;
        this.sobrenome = sobrenome;
    }

    public String getNomeCompleto() {
        return this.nome + this.sobrenome;
    }
}
```

1.2.3 Variáveis

Usando como exemplo a classe Pessoa definida anteriormente, nome e sobrenome são variáveis. A declaração de variáveis é bem simples, sempre o tipo seguido do nome da variável.

Dizemos que essas são variáveis de instância, pois existe uma cópia delas para cada objeto Pessoa criado em nosso programa. Cada cópia guarda o estado de uma certa instância desses objetos.

Existem ainda variáveis que não guardam valores ou referências para uma determinada instância, mas sim um valor compartilhado por todas as instâncias de objetos. Essas são variáveis estáticas, definidas com a palavra-chave ***static***. Veremos mais sobre esse tipo de membro mais à frente.

1.2.4 Métodos

A declaração de métodos é um pouquinho diferente pois precisamos do tipo do retorno, seguido do nome do método e seguido de parênteses, sendo que pode ou não haver parâmetros de entrada desse método. Cada parâmetro é uma declaração de variável em si. Essa linha do método, onde está definido o retorno, o nome e os parâmetros é onde temos a assinatura do método. Cuidado, pois a assinatura de um método inclui somente o nome do método e os tipos dos parâmetros.

Assim como variáveis, métodos também podem ser ***static***, como veremos mais adiante.

1.2.5 Construtores

Uma classe pode possuir zero ou vários construtores. Nossa classe Pessoa possui um construtor que recebe como parâmetros o nome e o sobrenome da pessoa. A principal diferença entre a declaração de um método e um construtor é que um construtor não tem retorno e possui o mesmo nome da classe.

1.2.6 Métodos com o mesmo nome da classe

Note que um construtor pode ter um ***return*** vazio:

```java
class X {
    int j = -100;

```

```
    X(int i) {
        if (i > 1)
        return;
        j = i;
    }
}
```

Caso o valor seja maior que ***1***, o valor de ***j*** será ***-100***, caso contrário, será o mesmo valor de ***i***.

1.2.7 Interfaces

Além de classes, também podemos declarar *interfaces* em nossos arquivos java. Para definir uma interface usamos a palavra reservada ***interface***:

```
interface Autenticavel {

    final int TAMANHO_SENHA = 8;

    void autentica(String login, String senha);
}
```

Em uma interface, devemos apenas definir a assinatura do método, sem a sua implementação. Além da assinatura de métodos, também é possível declarar constantes em interfaces.

1.2.8 Múltiplas estruturas em um arquivo

Em java, é possível definir mais de uma classe/interface em um mesmo arquivo java, embora devamos seguir algumas regras:

1. Podem ser definidos em qualquer ordem;
2. Se existir alguma classe/interface pública, o nome do arquivo deve ser o mesmo dessa classe/interface;
3. Só pode existir uma classe/interface pública por arquivo;
4. Se não houver nenhuma classe/interface pública, o arquivo pode ter qualquer nome.

Logo, são válidos:

```
// arquivo1.java
interface Bar {}

class Foo {}
```

```
// Foo.java
public class Foo {}

interface X {}
```

1.2.9 Pacotes e imports em arquivos com múltiplas estruturas

As regras de pacotes e ***imports*** valem também para arquivos com múltiplas estruturas definidas. Caso exista a definição de um pacote, ela vale para todas as classes/interfaces definidas nesse arquivo, e o mesmo vale para ***imports***.

Create executable Java applications with a main method; run a Java program from the command line; produce console output

Import other Java packages to make them accessible in your code

Compare and contrast the features and components of Java such as: platform independence, object orientation, encapsulation, etc.