ORACLE®

Oracle Press™

OCA/OCP Java® SE 7 Guia de Estudo do Programador I & II

(Exames 1Z0-803 & 1Z0-804)

Kathy Sierra
Bert Bates

ALTA BOOKS
E D I T O R A
Rio de janeiro, 2017

SUMÁRIO RESUMIDO

SUMÁRIO

Parte I

OCA e OCP

CAPÍTULOS

1 Declarações e Controle de Acesso

OBJETIVOS PARA A CERTIFICAÇÃO

- Identificadores e Palavras-Chave
- javac, java, main() e Imports
- Declarar Classes e Interfaces
- Declarar Membros de Classe
- Declarar Construtores e Arrays
- Criar Membros de Classe estáticos
- Usar enums

✓ Recapitulando

P&R Autoavaliação

Supomos que, como está pretendendo obter certificação, você já sabe o básico do Java. Se você for completamente iniciante na linguagem, este capítulo — e o resto do livro — vão ser confusos; então, certifique-se saber, pelo menos, o básico da linguagem antes de mergulhar neste livro. Dito isso, vamos começar com uma breve recapitulação de alto nível para colocá-lo no clima do Java, caso tenha estado longe por um tempo.

Recapitulação do Java

Um programa Java é em sua maior parte uma coleção de *objetos* falando com outros objetos através da invocação dos *métodos* um do outro. Cada objeto é de certo *tipo*, e esse tipo é definido por uma *classe* ou *interface*. A maioria dos programas Java usa uma coleção de objetos de muitos tipos diferentes. A seguir temos uma lista de alguns termos úteis para essa linguagem orientada para objetos (OO).

- **Classe** Um modelo que descreve os estados e comportamentos que objetos desse tipo suportam.
- **Objeto** Em runtime (tempo de execução), quando a Máquina Virtual Java (JVM – Java Virtual Machine) encontrar a palavra-chave `new`, ela vai usar a classe apropriada para criar um objeto que seja uma instância dessa classe. Esse objeto vai ter seu próprio estado e acesso a todos os comportamentos definidos pela sua classe.
- **Estado (variáveis de instância)** Cada objeto (instância de uma classe) vai ter seu próprio conjunto único de variáveis de instância, conforme definidas na classe. Coletivamente, os valores atribuídos às variáveis de instância de um objeto formam o estado do objeto.
- **Comportamento (métodos)** Quando um programador cria uma classe, ele cria métodos para essa classe. Os métodos são onde a lógica da classe é armazenada e onde o trabalho real é feito. Eles são onde os algoritmos são executados e os dados são manipulados.

Identificadores e Palavras-Chave

Todos os componentes do Java sobre o qual falamos — classes, variáveis e métodos — precisam de nomes. No Java, esses nomes são chamados de *identificadores* e, como seria de se esperar, há regras para o que constitui um identificador Java válido. Contudo, além do que é *válido*, os programadores Java (e Oracle) criaram *convenções* para a nomenclatura de métodos, variáveis e classes.

Como todas as linguagens de programação, o Java tem um conjunto interno de *palavras-chave* (*keywords*). Essas palavras-chave *não* podem ser usadas como identificadores. Mais adiante neste capítulo, vamos examinar os detalhes dessas regras e convenções de nomenclatura e das palavras-chave Java.

Herança

Central ao Java e a outras linguagens OO é o conceito de *herança*, que permite que código definido em uma classe seja reutilizado em outras classes. Em Java, você pode definir uma superclasse (mais abstrata) e então estendê-la com subclasses mais específicas. A superclasse não sabe nada das classes que herdam dela, mas todas as subclasses que herdam de uma superclasse devem declarar explicitamente a relação de herança. Uma subclasse que herde de uma superclasse recebe automaticamente métodos e variáveis de instância acessíveis definidos pela superclasse, mas a subclasse também é livre para sobrescrever os métodos da superclasse a fim de definir comportamento mais específico. Por exemplo, uma *super*classe `Car` poderia definir métodos gerais comuns a todos os automóveis, mas uma *sub*classe `Ferrari` poderia sobrescrever o método `accelerate()` que já havia sido definido na classe `Car`.

Interfaces

Um poderoso companheiro da herança é o uso das interfaces. Interfaces são como uma superclasse 100% abstrata que define os métodos que uma subclasse deve suportar, mas não *como* devem ser suportados. Em outras palavras, por exemplo, uma interface `Animal` pode declarar que todas as implementações de `Animal` devem ter um método `eat()`. Isso significa que é problema das classes que implementam a interface `Animal` definir o código real para como esse tipo em particular de `Animal` se comportará quando o método `eat()` for invocado.

Encontrando Outras Classes

Como veremos mais adiante neste livro (para os candidatos ao OCP), é uma boa ideia tornar suas classes *coesas*. Isso significa que cada classe deve ter um conjunto específico de responsabilidades. Por exemplo, se estivesse criando um programa de simulação de zoológico, você desejaria representar tamanduás com uma classe e os visitantes do zoológico com uma classe diferente. Além disso, você poderia ter uma classe `Zookeeper` e uma classe `PopcornVendor`. O ponto é que você não quer uma classe que tenha ambos os comportamentos de tamanduás e vendedores de pipoca (mais sobre isso no Capítulo 10).

Mesmo um programa Java simples usa objetos de muitas classes diferentes: algumas que *você* criou e algumas construídas por outros (como as classes da API Java da Oracle). O Java organiza as classes em *pacotes* e usa instruções *import* para proporcionar aos programadores uma maneira consistente de gerenciar a nomenclatura e o acesso às classes de que precisarem. O exame aborda *muitos* conceitos relacionados acesso a pacotes e classes; vamos explorar os detalhes ao longo do livro.

OBJETIVO PARA A CERTIFICAÇÃO

Identificadores e Palavras-Chave (Objetivos OCA 1.2 e 2.1)

1.2 Definir a estrutura de uma classe Java.

2.1 Declarar e inicializar variáveis.

Lembre-se de que, quando listamos um ou mais Objetivos de Certificação no livro, como acabamos de fazer, isso significa que a seção seguinte aborda pelo menos uma parte desse objetivo. Alguns objetivos serão abordados em diversos capítulos diferentes; então, você vai ver o mesmo objetivo em mais de um lugar no livro. Por exemplo, esta seção aborda declarações e identificadores, mas a *utilização* das coisas que você declara é abordada principalmente em capítulos mais à frente.

Então, vamos começar com os identificadores Java. Os dois aspectos dos identificadores que abordamos aqui são:

- **Identificadores Válidos** As regras que o compilador usa para determinar se um nome é válido.
- **Convenções de Código Java da Oracle** As recomendações da Oracle para a nomenclatura de classes, variáveis e métodos. Normalmente, obedecemos esses padrões ao longo do livro, exceto quando estamos tentando mostrar como uma questão capciosa do exame pode ser codificada. Você não vai ter que responder a questões sobre Convenções de Código Java, mas recomendamos veementemente que as utilize.

Identificadores Válidos

Tecnicamente, os identificadores válidos devem ser compostos apenas por caracteres Unicode, números, símbolo de cifrão e caracteres de conexão (como o sublinhado). O exame não entra em detalhes sobre quais intervalos do conjunto de caracteres Unicode são considerados qualificados como letras e números. Então, por exemplo, você não vai precisar saber que os dígitos tibetanos estão no intervalo `\u0420` a `\u0f29`. Aqui estão as regras que você *precisa* saber:

- Identificadores devem começar com uma letra, um caractere de cifrão (`$`) ou um caractere de conexão, como o sublinhado (_). Identificadores não podem começar com um número!
- Depois do primeiro caractere, os identificadores podem conter qualquer combinação de letras, símbolos de cifrão, caracteres de conexão ou números.

- Na prática, não há limite para o número de caracteres que um identificador pode conter.
- Você não pode usar palavras-chave Java como identificadores. A Tabela 1-1 lista todas as palavras-chave Java.
- Em Java, os identificadores diferenciam maiúsculas e minúsculas; `foo` e `FOO` são dois identificadores diferentes.

Seguem exemplos de identificadores válidos e inválidos. Primeiro, alguns identificadores válidos:

```
int _a;
int $c;
int ______2_w;
int _$;
int this_is_a_very_detailed_name_for_an_identifier;
```

Os seguintes são inválidos (é seu trabalho reconhecer o porquê):

```
int :b;
int -d;
int e#;
int .f;
int 7g;
```

Convenções de Código Java da Oracle

A Oracle estima que durante o tempo de vida de um trecho de código padrão, 20% do esforço são dispendidos na criação e teste originais do código e 80% do esforço são dispendidos na manutenção e no aprimoramento subsequentes do código

Manter-se de acordo e codificar para um conjunto de padrões de código ajuda a reduzir o esforço envolvido em testar, dar manutenção e aprimorar qualquer trecho de código. A Oracle criou um conjunto de padrões para o Java e publicou esses padrões em um documento inteligentemente intitulado "Java Code Conventions – Convenções de Código Java", que pode ser encontrado em `java.oracle.com`. É um documento excelente, curto e fácil de ler, e recomendamos muito sua leitura.

TABELA 1-1 Lista completa das palavras-chave do Java (`assert` adicionada no 1.4, `enum` adicionada no 1.5)

abstract	boolean	break	byte	case	catch
char	class	const	continue	default	do
double	else	extends	final	finally	float
for	goto	if	implements	import	instanceof
int	interface	long	native	new	package
private	protected	public	return	short	static
strictfp	super	switch	synchronized	this	throw
throws	transient	try	void	volatile	while
assert	enum				

Dito isso, você vai descobrir que muitas das questões do exame não seguem as convenções de código em decorrência das limitações no mecanismo de teste utilizado para disponibilizá-lo internacionalmente. Um dos grandes diferenciais da certificação Oracle é que os exames são administrados uniformemente no mundo todo. Para alcançar isso, as listagens de código que você vai ver no exame real frequentemente são comprimidas e não seguem os padrões de código da Oracle. Com a intenção de prepará-lo para o exame, vamos constantemente apresentar listagens de código que têm uma aparência "comprimida", frequentemente recuando nosso código com apenas dois espaços, ao contrário do padrão de quatro da Oracle.

Vamos também espremer nossas chaves de forma não natural e algumas vezes vamos colocar diversas instruções na mesma linha... ai! Por exemplo:

```
 1. class Wombat implements Runnable {
 2.   private int i;
 3.   public synchronized void run() {
 4.     if (i%5 != 0) { i++; }
 5.     for(int x=0; x<5; x++, i++)
 6.       { if (x > 1) Thread.yield(); }
 7.     System.out.print(i + " ");
 8.   }
 9.   public static void main(String[] args) {
10.     Wombat n = new Wombat();
11.     for(int x=100; x>0; --x) { new Thread(n).start(); }
12. } }
```

Considere-se avisado — você vai ver muitas listagens de código, questões simuladas e questões reais do exame deformadas. Ninguém quer que você escreva seu código assim — nem seu empregador, nem seus colegas de trabalho, nem nós, nem a Oracle e nem a equipe de criação do exame! Código como esse foi criado apenas para que conceitos complexos possam ser testados dentro de uma ferramenta universal de teste. Os únicos padrões que *são* seguidos tanto quanto possível no exame real são os de nomenclatura. Aqui estão os padrões de nomenclatura recomendados pela Oracle e que vamos usar no exame e na maior parte do livro:

- **Classes e Interface** A primeira letra deve ser maiúscula e se diversas palavras estiverem ligadas para formar o nome, a primeira letra das palavras interiores deve estar em maiúscula (um formato que é algumas vezes chamado de "CamelCase"). Para as classes, os nomes normalmente devem ser substantivos. Aqui estão alguns exemplos:

  ```
  Dog
  Account
  PrintWriter
  ```

  Para interfaces, os nomes normalmente devem ser adjetivos, como estes:

  ```
  Runnable
  Serializable
  ```

- **Métodos** A primeira letra deve estar em minúscula e então as regras normais do CamelCase devem ser usadas. Além disso, os nomes normalmente devem ser pares verbo–substantivo:

```
getBalance
doCalculation
setCustomerName
```

- **Variáveis** Como os métodos, o formato CamelCase deve ser usado, mas começando com uma letra minúscula. A Oracle recomenda nomes curtos e significativos que soem bem para nós. Alguns exemplos:

```
buttonWidth
accountBalance
myString
```

- **Constantes** As constantes Java são criadas marcando-se as variáveis como `static` e `final`. Elas devem ser nomeadas usando-se letras maiúsculas com o caractere sublinhado como separador:

```
MIN_HEIGHT
```

OBJETIVO PARA A CERTIFICAÇÃO

Definir Classes (Objetivos OCA1.2, 1.3, 1.4, 6.6 e 7.6)

1.2 Definir a estrutura de uma classe Java.

1.3 Criar aplicativos Java executáveis com um método main.

1.4 Importar outros pacotes Java para torná-los acessíveis no seu código.

6.6 Aplicar modificadores de acesso.

7.6 Usar classes e interfaces abstratas.

Quando você escreve código em Java, está escrevendo classes e interfaces. Dentro dessas classes, como você sabe, estão variáveis e métodos (e mais algumas coisas). O modo como você declara suas classes, métodos e variáveis afeta significativamente o comportamento do seu código. Por exemplo, um método `public` pode ser acessado por um código executado em qualquer lugar no seu aplicativo. Marque esse método como `private` e ele desaparece do radar de todo mundo (exceto da classe onde foi declarado).

Para este objetivo, vamos estudar as maneiras como você pode declarar e modificar (ou não) uma classe. Você vai descobrir que abordamos os modificadores

em um nível extremo de detalhe e apesar de sabermos que você já está familiarizado com eles, estamos começando do início. A maioria dos programadores Java acha que sabe como todos os modificadores funcionam, mas estudando mais de perto, frequentemente descobrem que não sabem (pelo menos não no grau necessário para o exame). Distinções sutis estão em todos os lugares, então você precisa estar absolutamente certo de estar completamente firme em tudo nos objetivos desta seção, antes de fazer o exame.

Regras de declaração de arquivo-fonte

Antes de mergulharmos nas declarações de classe, vamos dar uma revisada rápida nas regras associadas à declaração de classes, instruções `import` e instruções `package` em um arquivo-fonte.

- Só pode haver uma classe `public` por arquivo de código-fonte.
- Comentários podem aparecer no início ou no fim de qualquer linha do arquivo de código-fonte; eles são independentes de quaisquer das regras de posicionamento discutidas aqui.
- Se *houver* uma classe `public` em um arquivo, o nome do arquivo deve coincidir com o nome da classe `public`. Por exemplo, uma classe declarada como `public class Dog{}` deve estar em um arquivo de código-fonte chamado `Dog.java`.
- Se a classe faz parte de um pacote, a instrução `package` deve ser a primeira linha do arquivo de código-fonte, antes que quaisquer instruções `import` possam ser apresentadas.
- Se houver instruções `import`, elas devem estar entre a instrução `package` (se houver uma) e a declaração da classe. Se não houver uma instrução `package`, as instruções `import` devem ser as primeiras linhas do arquivo de código-fonte. Se não houver instruções `import` ou `package`, a declaração da classe deve ser a primeira linha do arquivo de código-fonte.
- Instruções `import` e `package` se aplicam a *todas* as classes dentro de um arquivo de código-fonte. Em outras palavras, não há maneira de declarar várias classes em um arquivo e tê-las em pacotes diferentes ou usar `imports` diferentes.
- Um aquivo pode ter mais de uma classe não-pública.
- Arquivos sem classes `public` podem ter um nome que não coincida com qualquer uma das classes no arquivo.

Usando os Comandos javac e java

Neste livro, vamos falar umas 1000 vezes sobre invocar os comandos `javac` e `java`. Apesar de, no **mundo real**, você provavelmente venha a usar um ambiente de