

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

Parte I

OCA e OCP

1	Declarações e Controle de Acesso	3
2	Orientação a Objetos	79
3	Atribuições	157
4	Operadores	211
5	Trabalhando com String, Arrays e ArrayLists	243
6	Controle de Fluxo e Exceções	289

Parte II

OCP

7	Assertivas e Exceções Java 7	353
8	Processamento de Strings, Formatação de Dados, Resource Bundles .	389
9	I/O e NIO	443
10	OO Avançado e Padrões de Projeto	503
11	Genéricos e Coleções	533
12	Classes Internas	633
13	Threads	63
14	Concorrência	729
15	JDBC	781
A	Sobre o CD	879
	Índice	885

SUMÁRIO

<i>Colaboradores</i>	v
<i>Agradecimentos</i>	xxv
<i>Prefácio</i>	xxvii
<i>Introdução</i>	xxix

Parte I

OCA e OCP

I Declarações e Controle de Acesso	3
Recapitulação do Java	4
Identificadores e Palavras-Chave (Objetivos OCA 1.2 e 2.1)	6
Identificadores Válidos	6
Convenções de Código Java da Oracle	7
Definir Classes (Objetivos OCA 1.2, 1.3, 1.4, 6.6 e 7.6)	9
Regras de declaração de arquivo-fonte	10
Usando os Comandos javac e java	10
Usando public static void main(String[] args)	12
Instruções Import e a API Java	13
Instruções Import Estáticas	14
Declarações e Modificadores de Classe	16
EXERCÍCIO 1-1 Criando uma Superclasse	
Abstrata e uma Subclasse Concreta	22
Usar Interfaces (OCA Objetivo 7.6)	23
Declarando uma Interface	23
Declarando Constantes de Interface	26
Declarar Membros de Classe	
(Objetivos OCA 2.1, 2.2, 2.3, 2.4, 2.5, 4.1, 4.2, 6.2 e 6.6)	27
Modificadores de Acesso	28
Modificadores de Não-Acesso a Membros	41
Declarações de Construtores	47
Declarações de Variáveis	48
Declarar e Usar enums (Objetivo OCA 1.2 e Objetivo OCP 2.5) ...	57
Declarando enums	58
✓ Recapitulando	65
P&R Autoavaliação	72
Respostas da Autoavaliação	77

2	Orientação a Objetos	79
	Encapsulamento (Objetivos OCA 6.1 e 6.7)	80
	Herança e Polimorfismo (Objetivos OCA 7.1, 7.2 e 7.3)	83
	É-UM	87
	TEM-UM	88
	Polimorfismo (Objetivos OCA 7.2 e 7.3)	92
	Sobrescrevendo/Sobrecarregando	
	(Objetivos OCA 6.1, 6.3, 7.2 e 7.3)	95
	Métodos Sobrescritos	96
	Métodos Sobrecarregados	101
	Casting (Objetivo OCA 7.3 e 7.4)	107
	Implementando uma Interface (Objetivo OCA 7.6)	111
	Tipos de Retorno Válidos (Objetivos OCA 2.2, 2.5, 6.1 e 6.3)	115
	Declarações de tipos de retorno	116
	Retornando um valor	117
	Construtores e Instanciação (Objetivos OCA 6.4, 6.5 e 7.5)	119
	Identificando Se um Construtor Padrão Será Criado	123
	Construtores Sobrecarregados	127
	Blocos de Inicialização	131
	Estática (Objetivo OCA 6.2)	133
	Variáveis e métodos estáticos	133
	✓ Recapitulando	142
	P&R Autoavaliação	147
	Respostas Da Autoavaliação	154
3	Atribuições	157
	Stack (Pilha) e Heap — Uma Revisão	158
	Literais, Atribuições e Variáveis	
	(Objetivos OCA 2.1, 2.2, 2.3 e Objetivo Atualizado 1.2)	160
	Valores Literais para Todos os Tipos Primitivos	160
	Operadores de Atribuição	164
	Exercício 3-1 Convertendo Tipos Primitivos	169
	Escopo (Objetivos OCA 1.1 e 2.5)	173
	Inicialização de Variáveis (Objetivo OCA 2.1)	176
	Usando uma Variável ou Elemento de Array	
	Que Não Tenha Sido Inicializado e Atribuído	176
	Tipos Primitivos e Objetos Locais (Pilha, Automáticos)	179
	Passando Variáveis Para Métodos (Objetivo OCA 6.8)	184
	Passando Variáveis de Referência de Objeto	185
	A Linguagem Java Usa a Semântica de Passagem Por Valor? ...	185
	Passando Variáveis Primitivas	186
	Coleta de Lixo (Objetivo OCA 2.4)	189
	Visão Geral do Gerenciamento da Memória e da Coleta de Lixo	189
	Visão Geral do Coletor de Lixo do Java	190

Escrevendo Código Que Torne os Objetos Explicitamente Qualificados	
Para a Coleta	192
Exercício 3-2 Experimento de Coleta de Lixo	197
✓Recapitulando	199
P&R Autoavaliação	202
Respostas da Autoavaliação	209
4 Operadores	211
Operadores Java (Objetivos OCA3.1, 3.2 e 3.3)	212
Operadores de Atribuição	212
Operadores Relacionais	214
Operador de Comparação instanceof	219
Operadores Aritméticos	222
Operador Condicional	227
Operadores Lógicos	227
✓Recapitulando	233
P&R Autoavaliação	235
Respostas da Autoavaliação	240
5 Trabalhando Com Strings, Arrays e ArrayLists	243
Usando String e StringBuilder (Objetivos OCA 2.7 e 2.6)	244
A Classe String	244
Fatos Importantes Relacionados aos	
Objetos String e à Memória	250
Métodos Importantes da Classe String	251
A Classe StringBuilder	254
Métodos Importantes da Classe StringBuilder	256
Usando Arrays (Objetivos OCA 4.1 e 4.2)	258
Declarando um Array	258
Construindo um Array	259
Inicializando Um Array	261
Usando ArrayList (Objetivo OCA 4.3)	272
Quando Usar ArrayLists	272
Métodos ArrayList em Ação	275
Métodos Importantes da Classe ArrayList	275
Encapsulamento para Variáveis de Referência	276
✓Recapitulando	279
P&R Autoavaliação	281
Respostas da Autoavaliação	287
6 Controle de Fluxo e Exceções	289
Usando as Instruções if e switch	
(Objetivos OCA 3.4 e 3.5 — e também Objetivo Atualizado 1.1) ..	290
Ramificação if-else	290
Instruções switch (OCA, OCP e Tópico Atualizado)	295

Exercício 6-1: Criando uma Instrução switch-case	302
Criando Loops (Objetivos OCA 5.1, 5.2, 5.3, 5.4 e 5.5)	302
Usando Loops while	302
Usando Loops do	303
Usando Loops for	304
Usando break e continue	309
Instruções Não Rotuladas	310
Instruções Rotuladas	311
Exercício 6-2: Criando um Loop while Rotulado	312
Tratando Exceções (Objetivos OCA 8.1, 8.2, 8.3 e 8.4)	313
Capturando uma Exceção Usando try e catch	313
Usando finally	315
Propagando Exceções Não Capturadas	317
Exercício 6-3: Propagando e Capturando uma Exceção	319
Definindo Exceções	320
Hierarquia de Exceções	321
Tratando uma Hierarquia de Classes de Exceções Inteira	322
Correspondência de Exceções	323
Declaração de Exceções e a Interface Pública	325
Relançando a Mesma Exceção	329
Exercício 6-4: Criando uma Exceção	330
Erros e Exceções Comuns (Objetivo OCA 8.5)	331
De Onde Vêm as Exceções	331
Exceções Lançadas pela JVM	332
Exceções Lançadas Programaticamente	332
Um Resumo das Exceções e Erros para o Exame	333
Fim da Parte I — OCA	333
✓TRecapitulando	337
P&R Autoavaliação	340
Respostas da Autoavaliação	349

Parte II

OCP

7 Assertivas e Exceções Java 7	353
Trabalhando Com o Mecanismo de Assertivas (Objetivo OCP 6.5) ..	354
Visão Geral das Assertivas	355
Ativando Assertivas	358
Usando as Assertivas Appropriadamente	361
Trabalhando Com o Tratamento de Exceções do Java 7 (Objetivos OCP 6.2 e 6.3)	364
Use a Instrução try Com Cláusulas multi-catch e finally	365

Recursos AutoCloseable Com a Instrução try-com-recursos	370
✓Recapitulando	378
Q&AAutoavaliação	380
Respostas da Autoavaliação	387
8 Processamento de Strings, Formatação de Dados, Pacotes de Recursos	389
String, StringBuilder e StringBuffer (Objetivo OCP 5.1)	390
Datas, Números, Moedas e Locales	
(Objetivos OCP 12.1, 12.4, 12.5 e 12.6)	390
Trabalhando com Datas, Números e Moedas	391
Parsing, Tokenização e Formatação (Objetivos OCP 5.1, 5.2 e 5.3)	402
Um Tutorial Sobre Busca	403
Encontrando Dados Via Combinação de Padrões	413
Tokenização	416
Formatando com printf() e format()	421
Pacotes de Recursos (Objetivos OCP 12.2, 12.3 e 12.5)	423
Pacotes de Recursos	423
Pacotes de Recursos de Propriedade	425
Pacotes de Recursos do Java	426
Localidade Padrão	426
Escolhendo o Pacote de Recursos Certo	427
✓ Recapitulando	431
P&RAutoavaliação	434
Respostas da Autoavaliação	441
9 I/O e NIO	443
Navegação em Arquivos e I/O (Objetivos OCP 7.1 e 7.2)	444
Criando Arquivos com a Classe File	446
Usando FileWriter e FileReader	448
Combinando Classes de I/O	450
Trabalhando com Arquivos e Diretórios	452
A Classe java.io.Console	456
Files, Path e Paths (Objetivos OCP 8.1 e 8.2)	457
Criando um Path	459
Criando Arquivos e Diretórios	461
Copiando, Movendo e Excluindo Arquivos	462
Recuperando Informações sobre um Path	464
Normalizando um Caminho	465
Resolvendo um Caminho	467
Relativizando um Path	468
Atributos de Arquivos e Diretórios (Objetivo OCP 8.3)	470
Lendo e Escrevendo Atributos da Maneira Fácil	470
Tipos de Interfaces de Atributos	471

Trabalhando com BasicFileAttributes	472
Trabalhando com DosFileAttributes	474
Trabalhando com PosixFileAttributes	475
Revisando os Atributos	475
DirectoryStream (Objetivo OCP 8.4)	476
FileVisitor (Objetivo OCP 8.4)	477
PathMatcher (Objetivo OCP 8.5)	481
WatchService (Objetivo OCP 8.6)	485
Serialização (Objetivo 7.2)	488
✓ Recapitulando	490
P&R Autoavaliação	492
Respostas da Autoavaliação	499
10 OO Avançado e Padrões de Projeto	503
É-UM e TEM-UM (Objetivos OCP 3.3 e 3.4)	504
Acoplamento e Coesão	505
Acoplamento	505
Coesão	506
Princípios de Composição de Objetos (Objetivo OCP 3.4)	507
Polimorfismo	509
Benefícios da Composição	510
Padrão de Projeto Singleton (Objetivo OCP 3.5)	510
O Que É um Padrão de Projeto	510
Problema	511
Solução	512
Benefícios	515
Padrão de Projeto DAO (Objetivo OCP 3.6)	515
Problema	516
Solução	517
Benefícios	520
Padrão de Projeto Factory (Objetivo OCP 3.7)	520
Problema	520
Solução	520
Benefícios	523
✓ Recapitulando	525
P&R Autoavaliação	527
Respostas da Autoavaliação	531
11 Genéricos e Coleções	533
toString(), hashCode() e equals() (Objetivos OCP 4.7 e 4.8)	534
O Método toString()	535
Sobrescrevendo equals()	536
Sobrescrevendo hashCode()	541
Visão Geral das Coleções (Objetivos OCP 4.5 e 4.6)	547
Então O Que Você Faz Com Uma Coleção?	547

Interfaces e Classes Chave do Framework Collections	548
Interface List	552
Interface Set	552
Interface Map	553
Interface Queue	554
Usando Coleções (Objetivos OCP 4.2, 4.4, 4.5, 4.6, 4.7 e 4.8)	556
Fundamentos do ArrayList	557
Autoboxing com Coleções	558
A Sintaxe “Diamante” do Java 7	561
Classificando Coleções e Arrays	562
Navegando (Buscando) em TreeSets e TreeMaps	576
Outros Métodos de Navegação	577
Backed Collections	578
Usando a Classe PriorityQueue e a Interface Deque	580
Visão Geral dos Métodos para Arrays e Collections	582
Visão Geral dos Métodos para List, Set, Map Queue	582
Tipos Genéricos (Objetivos OCP 4.1 e 4.3)	584
O Jeito Legado de Fazer Coleções	585
Genéricos e Código Legado	588
Polimorfismo e Genéricos	594
Métodos Genéricos	595
Declarações Genéricas	606
✓Recapitulando	614
P&R Autoavaliação	620
Respostas da Autoavaliação	629
12 Classes Internas	633
Classes Aninhadas (Objetivo OCP 2.4)	635
Classes Internas	635
Codificando uma Classe Interna “Regular”	636
Referenciando a Instância da Interna ou Externa Dentro da Classe Interna	639
Classes Internas Locais a Métodos	641
O Que Um Objeto Interno Local ao Método Pode e Não Pode Fazer	642
Classes Internas Anônimas	644
Boa e Velha Classe Interna Anônima, Tipo Um	644
Boa e Velha Classe Interna Anônima, Tipo Dois	647
Classes Internas Anônimas Definidas no Argumento	649
Classes Aninhadas Estáticas	650
Instanciando e Usando uma Classe Aninhada Estática	651
✓Recapitulando	653
P&R Autoavaliação	655
Respostas da Autoavaliação	661

13 Threads	663
Definir, Instanciar e Iniciar Threads (Objetivo OCP 10.1)	664
Definindo uma Thread	667
Instanciando uma Thread	668
Iniciando uma Thread	670
Estados e Transições de Threads	
(Objetivo OCP 10.2)	677
Estados das Threads	678
Evitando Execução de Threads	679
Sleeping	680
Exercício 13-1: Criando uma Thread e Colocando-a para Dormir	682
Prioridades de Thread e yield()	683
Sincronizando Código, Problemas com Threads	
(Objetivos OCP 10.3 e 10.4)	687
Sincronização e Locks	692
Exercício 13-2: Sincronizando um Bloco de Código	695
Impasse em Threads	701
Interação das Threads (Objetivos OCP 10.3 e 10.4)	702
Usando notifyAll() Quando Muitas Threads	
Podem Estar Esperando	707
✓ Recapitulando	712
P&RAutoavaliação	715
Resposta da Autoavaliação	725
Resposta dos Exercícios	728
14 Concorrência	729
Concorrência com o Pacote java.util.concurrent	730
Aplicar Variáveis Atômicas e Locks (Objetivo OCP 11.2)	730
Variáveis Atômicas	731
Locks	733
Usar Coleções java.util.concurrent	
(Objetivo OCP 11.1) e Usar uma Deque (Objetivo OCP 4.5)	740
Coleções de Cópia na Escrita	742
Coleções Concorrentes	743
Filas de Bloqueio	744
Usar Executors e ThreadPools (Objetivo OCP 11.3)	748
Identificando Tarefas Paralelas	749
Quantas Threads Você Pode Executar?	749
Tarefas Com Uso Intenso de CPU versus I/O	750
Lutando por um Turno	751
Desacoplando Tarefas das Threads	751
Usar o Framework Paralelo Fork/Join (Objetivo OCP 11.4)	757

Dividir e Conquistar	758
ForkJoinPool	759
ForkJoinTask	759
✓Recapitulando	770
P&RAutoavaliação	773
Respostas da Autoavaliação	779
15 JDBC	781
Começando: Uma Introdução aos Bancos de Dados e ao JDBC	782
Falando com um Banco de Dados	784
Bob's Books, Nosso Banco de Dados de Teste	787
Interfaces Centrais da API JDBC (Objetivo OCP 9.1)	790
Conectar um Banco de Dados Usando DriverManager	
(Objetivo OCP 9.2)	792
A Classe DriverManager	793
A URL JDBC	796
Versões da Implementação do Driver JDBC	798
Enviar Consultas e Ler Resultados do Banco de Dados	
(Objetivo OCP 9.3)	799
Todos os Clientes do Bob	799
Statements	801
ResultSets	805
Atualizando ResultSets (Não Está no Exame!)	824
Quando as Coisas Dão Errado – Exceções e Avisos (Warnings) ..	835
Usar Objetos PreparedStatement e CallableStatement	
(Objetivo OCP 9.5)	840
PreparedStatement	841
CallableStatement	843
Construir e Usar Objetos RowSet (Objetivo OCP 9.5)	846
Trabalhando com RowSets	848
Transações JDBC (Objetivo OCP 9.4)	853
Conceitos de Transação JDBC	855
Começando um Contexto de Transação no JDBC	855
Revertendo uma Transação	857
Usando Pontos de Salvamento com JDBC	858
✓Recapitulando	864
P&RAutoavaliação	867
Respostas da Autoavaliação	875
Sobre o CD	879
Índice	885



Parte I

OCA e OCP

CAPÍTULOS

- | | | | |
|---|----------------------------------|---|--|
| 1 | Declarações e Controle de Acesso | 5 | Trabalhando com Strings, Arrays e ArrayLists |
| 2 | Orientação a Objetos | 6 | Controle de Fluxo e Exceções |
| 3 | Atribuições | | |
| 4 | Operadores | | |



I

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 *convencões* 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 *superclasse* `Car` poderia definir métodos gerais comuns a todos os automóveis, mas uma *subclasse* `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				

8 Capítulo I: Declarações e Controle de Acesso

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 arquivo 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