Exercícios
para o
Exame de Certificação em Java

Capítulo 01

Questionário

1. Verdadeiro ou Falso: Um tipo de dados assinado tem um número igual de non-zero positivo e negativo estima disponível.

Falso. O alcance de número negativo é maior por 1 que o alcance de números positivos.

2. Escolha o identificadores válido desses listados abaixo.

A. Big01LongStringWiº
B. $int
C. bytes
D. $1
E. finalista

Todos os identificadores são válidos.

3. O qual das assinaturas seguintes são válidos para o principal () ponto de entrada de método de uma aplicação?

B. público arg de main(String nulo estático [])
D. público main(String nulo estático [] arg)

As alternativas B e D são ambas aceitáveis.

4. Se todos os três " elementos de topo-nível " acontecem em um arquivo de fonte, eles têm que aparecer em qual ordem?

A. Importações, declaração de pacote, classes,
B. Classes, importações, declarações de pacote,
C. Declaração de pacote tem que vir primeiro; ordene para importações e definições de classe não são significantes.
D. Empacote declaração, importações, classes,
E. Importações têm que vir primeiro; ordene para declaração de pacote e definições de classe não são significantes.

A alternativa D está correto. Esta ordem deve ser observada estritamente.

5. Considere a linha seguinte de código:

int x [] = new int[25];

Depois de execução que statment ou declarações são verdades?

A. x[24] é 0.
B. x[24] é indefinido.
C. x[25] é 0.
D. x[0] é nulo.
E. x.length é 25.

As alternativas A e E são verdades. O array tem 25 elementos, indexados de 0 por 24. Todos os elementos são inicializados com zero.

6. Considere a aplicação seguinte:

```
1.  class Q6
2.  { public static void main (String args [])
3.  { Holder h = new Holder();
4.  h.held = 100;
5.  h.bump(h);
6.  System.out.println(h.held);
7.  }
8.  }
9.
10. class Holder {
11. public int held;
12. public void bump(Holder theHolder)
13. {theHolder.held++;}
14. }
```

Que valor será impresso na linha 6?

A. 0
B. 1
C. 100
D. 101

A alternativa D está correta. Esta ordem deve ser estritamente observada.

7. Considere a aplicação seguinte:

```
1.  class Q7 {
2.  public static void main(String args[]) {
3.  double d = 12.3;
4.  Decrementer dec = new Decrementer();
5.  Dec.decrement(d);
6.  System.out.println(d);
7.  }
8.  }
9.
10. class Decrementer {
11. public void decrement(double decMe)
12. { decMe = decMe – 1.0;}
13. }
```

Que valor será impresso na linha 6?

A. 0.0
B. -1.0
C. 12.3
D. 11.3

A alternativa C está correta. O passado para o método decrement () uma cópia do argumento d; a cópia adquire decremento, mas o original está intacto.

Observe a diferença entre esta questão e a anterior:

Nesta questão o método Decrementer trabalha com uma cópia do parâmetro passado:

```
public void decrement(double decMe)
        {decMe = decMe – 1.0;}
```

Na questão anterior (6), o método trabalha diretamente com variável.

```
Public void bump(Holder theHolder)
        {theHolder.held++ ;}
```

8. Como você pode forçar o Garbage Collection de um objeto?

A. Garbae não podem ser forçados.
B. Chame System.gc ();
C. Chame System.gc (), passando em uma referência ao objeto a ser Garbage-collected.
D. Chame Runtime.gc ()
E. Fixe todas as referências para o objeto para valores novos (nulo, por exemplo).

A alternativa A está correta. Garbage colletion não pode ser forçado. Chamando System.gc () ou Runtime.gc () não é 100 por cento fidedigno, desde que a linha de Garbage-Collection poderia adiar a uma linha de prioridade mais alta; assim B e D estão incorretos. C está incorreto porque o dois métodos gc () não levam argumentos; de fato, isto você ainda tem uma referência para passar no método, o objeto não é contudo elegível ser colecionado. A alternativa E fará o objeto elegível para coleção o da próxima vez as corridas de coleção de lixo.

9. Qual é o alcance de valores que podem ser nomeados a uma variável do tipo short ?

A. Depende do hardware subjacente.
B. 0 até 2^{16} - 1
C. 0 até 2^{32} - 1
D. 2^{15} até 2^{15} - 1
E. 2^{31} até 2^{31} - 1

A alternativa D está correto. O alcance para um 16 bit pequeno é – 2^{15} por 2^{15} - 1. Este alcance é parte da especificação de java, embora o hardware subjacente.

10. Qual é o alcance de valores que podem ser nomeados a uma variável do tipo byte ?

A. Depende do hardware subjacente.
B. 0 por 2^{8} - 1
C. 0 por 2^{16} - 1
D. 2^{7} por 2^{7} - 1
E. 2^{15} por 2^{15} - 1

A alternativa D está correta. O alcance para uns 8-bit pequeno é –2^{7} por 2^{7} - 1.

Capítulo 02

1. Depois de execução do fragmento de código abaixo, quais são os valores das variáveis x, a e b ?

```
1. int x, a = 6, b = 7;
2. x = a++ + b++;
```

A. x = 15, a = 7, b = 8
B. x = 15, a = 6, b = 7
C. x = 13, a = 7, b = 8
D. x = 13, a = 6, b = 7

A alternativa C está correta. A declaração de tarefa é avaliada se era

x = a + b;
a = a + 1;
b = b + 1;

Então a tarefa para x que usa a soma de 6 + 7 = 13. Depois da adição, os valores de a e b são incrementados, são armazenados os novos valores, 7 e 8, nas variáveis.

2. Qual das expressões seguintes são legais? (Escolha uma ou mais)

A. int x = 6; x =!x;
B. int x = 6; if (! (x>3)) {}
C. int x = 6; x = ~x;

As alternativas B e C estão corretas. Em A o uso de ! é impróprio, desde que x é de tipo de int, não boolean. Este é um engano comum entre programadores de C e C++, desde que a expressão seria válida nesses idiomas. Na alternativa B, a comparação é deselegante (começa um incômodo equivalente de se (x <= 3), mas válido, desde a expressão (x> 3) é um tipo de boolean, e o operador ! pode ser aplicado corretamente a isto. Na alternativa C o operador de inversão de bitwise é aplicado a um tipo integrante. O padrão de pedaço de 6 olhares como 0 ...0110 onde a elipse representa 271 pedaços.

3. Qual dos resultados das seguintes expressões resultará em um valor positivo em x? (Escolhe uma).

A. int x = -1; x = x >>> 5;
B. int x = -1; x = x >>> 32;
C. byte x = -1; x = x >>> 5;
D. int x = -1; x = x >> 5;

A alternativa A está correta. Em todo caso, o padrão de bit para -1 é " todo os uns ". Na alternativa A isto é trocado à direita no máximo cinco lugares com a introdução de bits 0 de posições significantes. O resultado é 271 pedaços na posição menos significante do valor de int. Considerando que o bit mais significante é 0, isto representa um valor positivo (de fato 134217727). Na alternativa B o valor de troca é 32 bits. Isto resultará em nenhuma mudança para a variável x, desde que seja executado de fato por (32 mod 32) bits que é igual 0. Assim em B o valor de x está inalterado (- 1). Na alternativa C é realmente ilegal como o resultado de x >>> 5 é do tipo int, e não pode ser trocada para o tipo byte x sem declaração explícita. Até mesmo se o elenco foi somado e dá para byte x = -1; x = (byte) (x >> >5); o resultado da expressão x >>> seria calculada de 5 formas:

1. Primeiro promove x para o tipo int. Isto dá um resultado sinal-estendido que é um int -1 com 321 pedaços.

2. Executa a troca; isto se comporta da mesma forma do iten anterior, dando 134217727 que é o valor de 271 bits int das posições menos significantes.

3. Trocando o resultado da expressão simplesmente "cortar", os oito bits menos significantes, desde que todos estes sejam "uns", o byte resultante representa -1.

Finalmente, a alternativa D executa uma troca assinada que propaga 1 bits na posição mais significante. Assim, neste caso, o valor resultante de x está inalterado a - 1.

4. Qual das expressões seguintes são legais? (Escolhe uma ou mais).

A. String x = Oi "; int y = 9; x + = y;
B. String x = Oi "; int y = 9; se (x == y) {}
C. String x = Oi "; int y = 9; x = x + y;
D. String x = Oi "; int y = 9; y = y + x;
E. String x = nulo;
int y = (x! = null) && (x.length ()> 0)? x.length (): 0;

As alternativas A, C e E são todas legais. Na alternativa A o uso de + = é tratado como uma taquigrafia para a expressão em C. Isto tenta "somar " um int para uma String que resulta em conversão do int para o tipo String -" 9 " neste caso - há a concatenação dos dois objetos para String. Assim neste caso, é executado o valor de x depois do código é " Hello9 ".

Na alternativa B a comparação (x == y) não é legal, porque apenas o operador + executa conversão implícita para um objeto String. A y variável é do tipo int e não pode ser comparado com um valor de referência. Não esqueça daquela comparação == que usa nos testes os valores e que para objetos, o " valor " é o valor de referência e não os conteúdos.

A alternativa C é idêntica a alternativa A sem o uso do operador de tarefa de taquigrafia.

A alternativa D calcula y + x que é legal em si mesmo porque produz uma String da mesma maneira como fez x + y. Ela tenta nomear o resultado que é " 9Hello " em uma variável de int então. Como o resultado de y + x é uma String, isto não é permitido.

A alternativa E é bastante diferente das outras. Os pontos importantes são o uso de curto circuito o operador && e o operador ternário?:. O operador à esquerda do && é o operador que sempre é avaliado, e neste caso a condição (x != nulo) é falsa. Porque isto é falso, a parte da mão direita da expressão (x.length ( )> 0) não precisa ter sido avaliada, porque o resultado do operador && é conhecido para ser falso. Iste pequeno-cirucuito é efetue nitidamente e evita a execução da chamada de método x.length () que falharia com um NullPointerException no momento da sua corrida. Este resultado falso é então usado na avaliação da expressão ternária. Como o valor de boolean é falso, o resultado da expressão global volta à direita o valor do cólon que é 0.

5. Qual dos fragmentos de código seguintes compilaria prosperamente e imprimiria " Iguale " quando corre? (Escolha uma ou mais)

A.
```
int x = 100; float y = 100.0F;
if (x == y) {System.out.println(“Equal ”);}
```

B.
```
int x = 100; Interger y = new Interger(100);
if (x == y) {System.out.println(“Equal ”);}
```

C.
```
Integer x = new Integer(100);
Integer y = new Integer(100);
if (x == y) {System.out.println(“Equal ”);}
```

D.
```
String x = new String(“100”);
String y = new String(“100”);
if (x == y) {System.out.println(“Equal ”);}
```

E.
```
String x = “100”;
String y = “100”;
if (x == y) {System.out.println(“Equal ”);}
```

As alternativas A e E estão corretas. Embora int e float não sejam compatíveis, eles geralmente podem ser misturados em qualquer lado de um operador. Desde então == não é nenhuma tarefa mas do que um operador de comparação, isto simplesmente causa promoção normal, de forma que o int é promovido a um valor do tipo float 9.0 e é comparado

prosperamente com o outro float de valor 9.0F. Por isto alternativa A é verdade.
O código em na alternativa B na verdade não compila. Isto é por causa do mismatch entre o tipo int e objeto de Inteiro. O valor de um objeto é sua referência, e nenhuma conversão sempre é possível entre referências e tipos numéricos. Isto aplica a qualquer conversão, não só compatibilidade de tarefa.
Na alternativa C, compila o código prosperamente, desde que a comparação esteja entre duas referências de objeto. Porém, o teste para igualdade compara o valor das referências (o endereço de memória tipicamente) e desde as variáveis x e y se referem a dois objetos diferentes, o teste devolve falso. O código na alternativa D se comporta do mesmo modo exatamente.
Comparando as alternativas E com a D poderia o persuadir que a alternativa E não deveria imprimir " Igual " provavelmente. De fato faz assim por causa de um optimizador exigido. Considerando que os objetos da String são imutáveis, o literal é inevitavelmente fio constante, assim o compilador ré-usa o mesmo objeto da String tendo-se o mesmo valor literal acontecendo-se mais de uma vez na fonte. Isto significa que as variáveis que x e y se referem de fato ao mesmo objeto; assim o teste (x ==y) é verdade e a " mensagem Igual " é impressa. É particularmente importante que você não permita este comportamento especial para o persuadir que o operador == possa ser usado para comparar os conteúdos de objetos de qualquer modo geral.

6. Qual é o resultado obtido após a execução do seguinte código:

```
1. public class Short {
2. public static void main(String args[ ]) {
3. StringBuffer s = new StringBuffer(“Hello”);
4. If ((s.length( ) > 5 ) &&
5.       (s.append(“ there”).equals(“False”)));
6. // do nothing
7. System.out.println(“value is “ + s);
8. }
9. }
```

A. O valor de saída é Hello.

B. O valor de saída é Hello there.

C. Irá ocorre um erro de compilação na linha 4 or 5.

D. Não haverá valor de saída.

E. Ocorrerá um NullPointerException.

A alternativa A está correta. O efeito do operador && é primeiro avaliar o operando à esquerda. Isso é a expressão (s.length( )> 5). Como o tamanho do objeto de StringBuffer s é de fato 5, este teste devolve falso. Usando a identidade lógica falso AND X = falso, o valor do condicional global é completamente determinado, e o && o operador salta avaliação do operando da mão direita então. Como resultado, o valor no objeto de StringBuffer ainda é simplesmente " Oi " quando for impresso.

Se o teste no lado à esquerda de && tinha devolvido verdadeiro, como teria acontecido tido o StringBuffer contido um segmento de texto mais longo, então o lado da mão direita teria sido avaliado. Embora poderia parecer um pouco estranho, aquela expressão, (s.append (lá ").equals (false ")), é válido e retorna um boolean. De fato, é garantido o valor da expressão ser falso, desde que é claramente impossível para qualquer StringBuffer conter precisamente " Falso " quando teve há pouco uma String " lá " juntado a isto. Isto é porém irrelevante - a essência desta expressão é que, se é avaliado, tem o efeito colateral de mudar o StringBuffer original juntando o texto " lá ".

7. Qual é o resultado obtido após a execução do seguinte código:

```
1. public class Xor {
2. public static void main(String args[ ]) {
3. byte b = 10; // 00001010 binary
4. byte c = 15; // 00001111 binary
5. b = (byte) (b ^ c);
6. System.out.println(“b contains “ + b);
7. }
8. }
```

A. O valor de saída é b = 10.

B. O valor de saída é b = 5.

C. O valor de saída é b = 250.

D. O valor de saída é b = 245.

A alternativa B está correta. O uso exclusivo do operador or ^ trabalha nos pares de pedaços em posição equivalente no dois operadores. Neste exemplo produz isto:

00001010
00001111

XOR ----------

00000101

A advertência é que os únicos 1 bits na resposta estão nessas colunas onde exatamente um do operandos tem um 1 bit. Se nenhum, ou ambos, os operandos tem um 1, então os resultados serão 0 bit.

O valor 00000101 binário corresponde a 5 decimal.

É preço que se lembra que, embora este exemplo foi mostrado como um cálculo de byte, o funcionamento atual é int usando acabado (32-bit) valores. Isto é por que o elenco explícito é requerido antes do resultado é nomeado no b variável em linha 5.

8. Qual o resultado obtido após a execução do seguinte código ?

```
1. public class Ternary {
2. public static void main(String args[ ]) {
3. int x = 4;
4. System.out.println("value is " +
5. (( x > 4) ? 99.99 : 9 ));
6. }
7. }
```

A. O valor de saída é 99.99.

B. O valor de saída é 9.

C. O valor de saída é 9.0.

D. Ocorrerá um erro de compilação na linha 5.

A alternativa C está correta. Neste código os valores de resultado opcionais para o operador ternário, 99.99 (um double) e 9 (um int), são de tipos diferentes. O tipo de resultado de um operador ternário deve ser determinado completamente em tempo de compilação, e neste caso o tipo escolhido é o de usar as regras de promoção para operandos binário, que o é tipo double. Porque o resultado é um double, o valor de produção está impresso em um formato de ponto flutuante.

A escolha do qual os dois valores para produção é feita em base do valor de boolean que precede o ?. Desde que x é 4, o teste (x> 4) é falso. Isto causa a expressão global para levar o segundo dos possíveis valores que são 9 em lugar de 99.99. Porque o tipo de resultado é promovido para um tipo double, o valor de produção é escrito de fato como 9.0 em lugar de o mais óbvio 9.

Se os dois possíveis tipos de argumento tivessem sido completamente incompatíveis, por exemplo (x> 4)? " Oi ": 9, então o compilador teria emitido um erro naquela linha.

9. Qual o valor obtido após a passagem pelo fragmento de código a seguir ?

```
1. int x = 3; int y = 10;
2. System.out.println(y % x);
```

A. 0.

B. 1.

C. 2.

D. 3.

A alternativa B mostra a saída correta. Neste caso, o cálculo é relativamente direto desde que inteiro só positivo seja envolvido. Dividindo 10 por 3 dá 3 sobrando 1, e este 1 forma o resultado da expressão de módulo. Outro modo para pensar neste cálculo é 10 - 3 = 7, 7 - 3 = 4, 4 - 3 = 1, 1 é então menos que 3 o resultado é 1. A segunda aproximação é realmente mais geral, desde que dirige cálculos de flutuante-ponto, também. Não esqueça que para números de negativo, você deveria ignorar os sinais durante o cálculo separe, e simplesmente prenda o sinal do operando à esquerda para o resultado.

10. Qual o resultado obtido após a execução do fragmento de código abaixo ?

```
1. int x = 1;
2. String [ ] names = { “Fred”, “Jim”, “Sheila” };
3. names[--x] += “.”;
4. for (int i = 0, i < names.length; i++) {
5. System.out.println(names[i]);
6. }
```

A. O valor de saída será Fred.

B. O valor de saída será Jim.

C. O valor de saída será Sheila.

D. Não haverá nenhum valor de saída.

E. Ocorrerá um erro do tipo ArrayIndexOutOfBoundsException.

A alternativa A está correta. Os operadores de tarefa do op de forma = só avalie a expressão à esquerda uma vez. Assim o efeito de decrementing x, em--x, só acontece uma vez e resulta em um valor de 0 e não -1. Então nenhum fora-de-salto forma são tentados acessos. Os elementos de ordem que são afetado por isto operações são " Fred ", desde que o decrement acontece antes o + = operação é executada. Embora Fio objeta que eles são immutable, as referências que são os elementos de ordem não são. É completamente possível causar o valor name[0] ser modificado para se referir a um Fio recentemente construído que acontece para ser " Fred ".

Capítulo 03

1. Qual das seguintes declarações são ilegais ?

   A. friendly String s;

   B. transient int i = 41;

   C. public final static native int w( );

   D. abstract double d;

   E. abstract final double hyperbolicConsine( );

A, D, e E são ilegais. A alternativa A é ilegal porque "amigável" não é um Keyword.A alternativa B é uma declaração passageira legal. A alternativa C é estranha mas legal. A alternativa D é ilegal porque só métodos e classes podem ser abstratas. A alternativa E é ilegal porque abstrato e final é contraditório.

2. Qual das declarações é verdadeira ?

   A. Uma classe abstract não pode ter nenhum método final.

   B. Uma classe final não pode ter nenhum método abstract.

A alternativa B é verdadeira: Uma classe final pode não ter nenhum método abstrato. Qualquer classe abstrata, métodos devem ser abstrato, e uma classe pode não ser do tipo abstrato e final. A declaração da alternativa A diz que na classe abstrata pode não ter métodos finais, mas não há nada errado com isto. A classe abstrata vai ser sub-classe eventualmente, e a subdivisão de classe tem que evitar anular os métodos finais do pai. Qualquer outro método pode ser anulado livremente.

3. Qual será a modificação mínima que permitirá a compilação do código a seguir ?

```
1. final class Aaa
2. { int xxx;
3.     void yyy( ) { xxx = 1; }
4. }
5.
6. class Bbb extends Aaa
```

```
7.  { final Aaa finalref = new Aaa( );
8.  final void yyy ( )
9.  { System.out.println("In method yyy( )");
10. finalref.xxx = 12345;
11. }
12. }
```

A. Na linha 1, remova o modificador final.

B. Na linha 10, remova o modificador final.

C. Remova a linha 15.

D. Nas linha 1 e 10, remova os modificadores final.

E. O código irá compilar. Não será necessário nenhuma modificação.

A alternativa A é a resposta correta. O código não compilará porque na linha 1 a classe Aaa é declarada final e pode não ser subclasseada. Aslinhas 10 e 15 estão corretas. A instância finalref variável é final, assim pode não ser modificada; enlata só referência que o objeto criou em linha 10. Porém, o dados dentro daquele objeto não é final, assim não há nada errado com linha 15.

4. Qual das seguintes declarações é a verdadeira ?

A. Podem não ser anulados métodos passageiros.

B. Devem ser anulados métodos passageiros.

C. Podem não ser seriadas classes passageiras.

D. Variáveis passageiras devem ser estáticas.

E. Variáveis passageiras não são seriadas.

A alternativa E está correta. As alternativas A, B, e C não querem dizer nada, porque só variáveis podem ser passageiras, não métodos ou classes. A alternativa D é falsa porque variáveis passageiras nunca podem ser estáticas. A alternativa E é uma boa um-setence definição de visitante.

5. Qual das declarações na aplicação abaixo é verdadeira ?

```
1. class StaticStuff
2. { static int x = 10;
3. static { x += 5; }
```

```
4. public static void main(String args[ ])
5. {
6.  System.out.println("x =  " + x);
7. }
8. static {x /= 5;} }
```

A. As linhas 5 e 12 não irão compilar, devido a falta dos nomes dos métodos e dos tipos de retorno.

B. A linha 12 não irá compilar, porque você não pode chamar apenas o inicializador static.

C. O código irá compilar e a saída será x = 10.

D. O código irá compilar e a saída será x = 15.

E. O código irá compilar e a saída será x = 3.

A alternativa E está correta. Inicializadores estáticos múltiplos (enfileira 5 e 12) são permitidos. Todo o código do inicializador estático é executado em momento de classe-carga, assim antes de principal () sempre seja corrido, os dividiram por 5 (linha 12).

6. Qual das declarações abaixo é verdadeira ?

```
1. class HasStatic {
2. private static int x = 100;
3. public static void main(String args[ ])
4. { HasStatic hs1 = new HasStatic( );
5. hs1.x++;
6. HasStatic hs2 = new HasStatic( );
7. hs2.x++;
8. hs1 = new HasStatic( );
9. hs1.x++;
10. HasStatic.x++;
11. System.out.println("x = " + x);
12. } }
```

A. A linha 8 não irá compilar, porque ela possui uma referência static para uma variável private.

B. A linha 13 não irá compilar, porque ela possui uma referência static para uma variável private.

C. O programa irá compilar e a saída será x = 102.

D. O programa irá compilar e a saída será x = 103.

E. O programa irá compilar e a saída será x = 104.

A alternativa E está correta. O programa compila bom; o "referência estática para uma variável privada" materiais em respostas UM e B é tolice. O x variável estático é incrementado quatro vezes, em linhas 8,10,12,and13.

7. Dado o código abaixo, e fazendo nenhuma outra mudança que tem acesso modificadores (o public, protected, ou private) pode ser colocado legalmente antes de aMethod () na linha 3? Se a linha 3 permanece como esta, qual palavras reservada podem ser colocados legalmente antes de aMethod () na linha 8?

```
1. class SuperDuper {
2. void aMethod( ) { }
3. }
4. class Sub extends SuperDuper {
5. void aMethod( ) { }
6. }
```

Na linha 3, o método pode ser declarado privado. O acesso de método da versão de subdivisão de classe (linha 8) é amigável, e só um método privado ou amigável pode ser anulado para ser amigável. O princípio básico é que um método pode não ser anulado para ser mais privado. (Veja Figura 3.2) Em linha 8 (linha 3 pretensiosa permanece só), a versão de suprclass é amigável, assim a versão de subdivisão de classe pode estar como é (e é amigável), ou pode ser declarado protegido ou público.

8. Deverão ser usados qual modificador ou modificadores para denotar uma variável que não deveria ser escrita fora como parte de seu estado de presistente de classe?

A. private

B. protected

C. private protected

D. transient

E. private transient

A resposta correta é D (transitent). Os outros modificadores controlam acesso de outros objetos dentro do Java Máquina Virtual.

Responda para E (o Visitante privado) também trabalha mas não é mínimo.

As próximas 02 questões serão baseadas na definição de classe a seguir:

```
1. package abcde;
2. public class Bird {
3. protected static int referenceCount = 0;
4. public Bird( ) {referenceCount++;}
5. protected void fly( ) {/*Flap wings, etc. */}
6. static int getRefCount( ) {return referenceCount;}
7. }
```

9. Qual das declarações nas classes Bird e Parrot é verdadeira ?

```
1. package abcde;
2. class Parrot extends abcde.Bird {
3. public void fly( ) {/*Parrot specific flight code.*/}
4. public int getRefCount( ) {return referenceCount}
5. }
```

A. Compilação do Parrot.java falha na linha 4, porque o método fly() é protegido na superclass e a classes Bird e Parrot estão no mesmo pacote.

B. Compilação do Parrot.java falha na linha 4, porque o método fly() é protegido na superclass e público na subdivisão de classe e não podem ser anulados métodos para ser mais públicos.

C. Compilação de Parrot.java falha na linha 5, porque o método getRefCount( ) é estático na superclass e os métodos estáticos não podem ser overriden para ser non-estáticos.

D. Compilação de Parrot.java terá sucesso, mas uma exceção de runtime é lançada se o método fly( ) for sempre chamado em uma instância da classe Parrot.

E. Compilação de Parrot.java tem sucesso, mas uma exceção de runtime é lançada se o métodogetRefCount( ) for sempre chamado em uma instância da classe Parrot.

A alternativa C está correta: Podem não ser anulados métodos estáticos para ser non-estáticos. B está incorreto porque declara o caso para trás: Métodos na verdade podem ser mais privados, não mais público. Respostas A, D, e E não fazem sentido nenhuma.

10. Qual das declarações nas classes Bird e Nightingale abaixo é verdadeira ?

```
1. package singers;
2.
3. class Nightingale extends abcde.Bird {
4. Nightingale( ) { referenceCount++;}
5.
6. public static void main(String args[ ]) {
7. System.out.println(“BEFORE: “ + referenceCount);
8. Nightingale florence = new Nightingale( );
9. System.out.println(“ AFTER: “ + referenceCount);
10. florence.fly( );
11. } }
```

A. O programa irá compilar e executar. A saída será Before: 0 e After: 2.

B. O programa irá compilar e executar. A saída será Before: 0 e After: 1.

C. A compilação do Nightingale irá falhar na linha 4, porque statics não podem ser anulados.

D. A compilação do Nightingale irá falhar na linha 10, porque o método fly( ) é protegido na super-classe.

E. A compilação do Nightingale irá ser bem sucedida, mas uma exception irá ocorrer na linha 10, porque o método fly( ) é protegido na super-classe.

A alternativa A está correta. Não há nada errado com Rouxinol. O referenceCount estático é batido duas vezes: uma vez em linha 4 de Rouxinol, e uma vez em linha 5 de Pássaro. (O constructor de nenhum-argumento do superclass sempre é caled de implicitly no começo de uma classe constructor de ', a menos que um constructor de superclass diferente seja pedido. Isto não tem nada que ver Com o tópico deste chapter,but é coberto em Capítulo 6, Objetos e Classes.) ReferenceCount de Sinse é batido duas vezes e não só uma vez, resposta B está errado. C diz que não podem ser anulados atatics, mas nenhum método estático está sendo anulado em linha 4; tudo aquilo está acontecendo é incre-ment de na de na herdaram variável estática. D está errado, desde protegeu é justamente o modificador de acesso nós queremos Pássaro. Voe () ter: Nós estamos chamando Pássaro. Voe () de uma subdivisão de classe em um pacote diferente. Resposta E é ridículo, mas usa terminologia acreditável.

Capítulo 04

1. Qual das seguintes declarações é a correta?

A. São convertidos automaticamente somente tipos primitivos, para mudar o tipo de uma referência de objeto, você tem que fazer um cast.

B. São convertidas automaticamente somente referências de objeto; mudar o tipo de um primitivo, você tem que fazer um cast.

C. Promoção de aritmética de referências de objeto requer arremesso explícito.

D. Primitivos e referências de objeto podem ser ambos convertidos e podem fazer cast.

E. Casting de tipos numéricos podem requerer um cheque de runtime.

A alternativa D está correta. C está errado porque objetos não levam parte em operações de aritmética. E está errado porque lançando potencialmente só de referências de objeto requer um cheque de runtime.

2. Qual das linha a seguir não irá compilar ?

```
1. byte b = 5;
2. char c = '5';
3. short s = 55;
4. int i = 555;
5. float f = 55.5F;
6. b = s;
7. i = c;
8. if (f > b)
9.   f = i;
```

Enfileire 6 (b = s) não compilará porque convertendo um pequeno a um byte é uma conversão de estreitamento que requer elenco de anexplicit. As outras tarefas no código estão alargando conversões.

3. O código abaixo irá compilar ?

```
1. byte b = 2;
2. byte b1 = 3;
3. b = b * b1;
```

O código surpreendentemente, não compilará a linha 3. São convertidos o dois operands que são originalmente bytes a ints antes da

multiplicação. O resultado da multiplicação é int de na que não pode ser nomeado a byte B.

4. No código abaixo, quais são os possíveis tipos para resultado de variável?

1. byte b = 11;
2. short s = 13;
3. result = b * ++s;

A. byte, short, int, long, float, double.

B. Boolean, byte, short, char, int, long, float, double.

C. Byte, short, char, int, long, float, double

D. Byte, short, char

E. Int, long, float, double

A alternativa E está correta. O resultado do cálculo em linha 2 é int de na (porque todos os resultados de aritmética são ints ou mais largo). Int de Na podem ser assingned a int de na, longo, flutue, ou dobre.

5. Considere a seguinte class:

```
1. class Cruncher {
2. void crunch(int i) {System.out.println(“int version ”);}
3. void crunch(String s) {System.out.println(“String version ”);}
4.
5. public static void main(String args[  ]) {
6. Cruncher crun = new Cruncher( );
7. char ch = ‘p’;
8. crun.crunch(ch);
9. }
10.}
```

Qual das seguintes declarações abaixo é verdadeira ?

A. A linha 3 não irá compilar, porque o método void não pode ser anulado.

B. A linha 8 não irá compilar, porque não existe uma versão de crunch( ).

C. O código irá compilar mas irá ocorre uma execption na linha 8.

D. O código irá compilar e a procedure terá como saída: int version

E. O código irá compilar e a procedure terá como saída: String version

A alternativa D está correta. A linha 8, o ch de argumento de serviço doméstico é alargado para digitar int (uma conversão de methodcall), e passou à versão de int de ruído de método ().

6. Qual das declarações abaixo é verdadeira ?

A. Referencias de objetos podem ser convertidas em tarefas mas não em chamadas de método.

B. Referencias de objete podem ser convertidas em chamadas de método mas não em tarefas.

C. Referencias de objete podem ser convertidas em ambos os métodos, chamadas e tarefas, mas as regras que governam estas conversões são muito diferentes.

D. Referencias de objete podem ser convertidas em ambos os método, chamadas e tarefas, e as regras que governam estas conversões são idênticas.

E. Referencias de objetos nunca podem ser convertidas.

A alternativa D está correta.

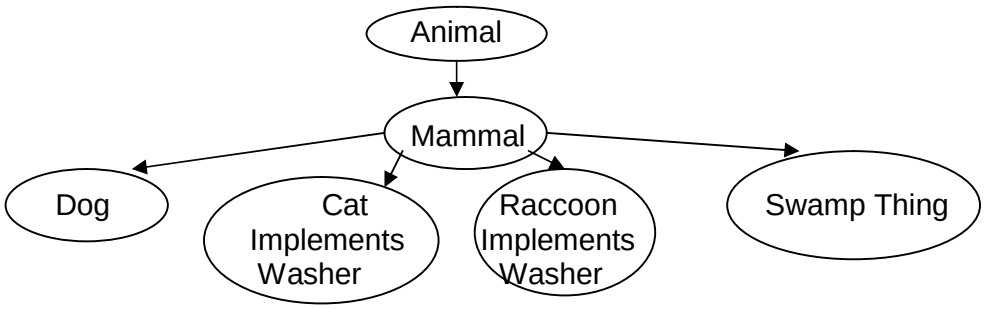
7. Considere o seguinte código:

```
1. Object ob = new Object ( );
2. String stringarr[ ] = new String[50];
3. Float floater = new Float(3.14F);
4.
5. ob = stringarr;
6. ob = stringarr[5];
7. floater = ob;
8. ob = floater;
```

Qual destas linhas acima não compilará ?

Enfileire 7 não compilarão. Na variável Objetam a Floast vai "abaixo" a árvore de hierarquia de herança, assim na que elenco explícito é requerido.

As questões 8, 9 e 10 referem-se a hierarquia de class mostrada na figura abaixo:



8. Considere o código abaixo:

1. Dog rover, fido;
2. Animal anim;
3.
4. rover = new Dog( );
5. anim = rover;
6. fido = new rover;
7. fido = (Dog)anim;

Qual das declarações abaixo é verdadeira ?

A. A linha 5 não irá compilar.

B. A linha 6 não irá compilar.

C. O código irá compilar mas irá ocorrer uma exception na linha 6.

D. O código irá compilar e rodar.

E. O código irá compilar e rodar, mas não é necessário o cast da linha 6 e pode ser eliminado.

A alternativa D está correta. O código compilará e correrá; o elenco em linha 6 é requerido, porque Animal de na variável para um Cachorro vai "abaixo" a árvore.

9. Considere o código abaixo;

1. Cast sunflower;
2. Washer wawa;
3. SwampThing pogo;
4.
5. sunflower = new Cat( );
6. wawa = sunflower;

7. pogo = (SwampThing)wawa;

Qual das declarações abaixo é verdadeira ?

A. A linha 6 não irá compilar; um cast explicito é necessário para converter o Cat para o Washer.

B. A linha 7 não irá compilar, porque você não pode fazer um cast da interface para a classe.

C. O código irá compilar e rodar, mas o cast da linha 7 não é necessário e pode ser eliminado.

D. O código irá compilar mas irá ocorrer uma exception na linha 7, porque a conversão em tempo de execução da interface para a classe não é permitida.

E. O código irá compilar mas irá ocorrer uma exception na linha 7, porque em tempo de execução a classe wawa não pode ser convertida para o tipo SwampThing.

A alternativa E está correta. O elenco na linha 7 é requerido. Resposta D é uma declaração prepóstera expressada em um tom de autoridade.

10. Considere o código a seguir:

```
1. Raccon rocky;
2. SwampThing pogo;
3. Washer w;
4.
5. rocky = new Raccon( );
6. w = rocky;
7. pogo = w;
```

Qual das declarações a seguir é verdadeira ?

A. A linha 6 não irá compilar; um cast explicito se faz necessário para converter um Raccon para um Washer.

B. A linha 7 não irá compilar; um cast explícito se faz necessário para converter um Washer para um SwampThing.

C. O código irá compilar e rodar.

D. O código irá compilar mas irá ocorrer uma exception na linha 7, porque em tempo de execução uma conversão da interface para uma classe não é permitida.

E. O código irá compilar mas irá ocorrer uma exception na linha 7, porque em tempo de execução a classe w não pode ser convertida para o tipo SwampThing.

A alternativa B está correta. A conversão em linha 6 está bem (classe para conectar), mas a conversão em linha 7 (interface para classificar) não é permitido. Um elenco em linha 7 fixará o problema.

Capítulo 05

1.

As alternativas B, C, D, e F estão corretas. O dá laçada interate i de 0 a 1 e j de 0 a 2. Porém, o interno dê laçada executa um continua declaração sempre que os valores de i e j são o mesmo. Desde que a produção é gerada dentro o interno dê laçada, depois de continuar declarando, isto significa que nenhuma produção é gerada quando os valores são o mesmo. Então, as produções sugeridas por respostas UM e são saltados E.

2.

A alternativa D está correta. Os valores de i aparecem fixos para levar os valores 0 e 1 para cada destes valores, j levam estima 0, 1 e 2. Howevwer, sempre que i e j têm o mesmo valor, o exterior dê laçada é continuado antes da produção é gerado. Desde o exterior dê laçada é o objetivo de continua declaração, o todo do interno dê laçada é abandonado. A única linha para ser produção é isso mostrado em D como a condição começando, i = 0 e j = 0 causam i imediatamente para assumir o valor 1, e assim que sejam fixados i e j a 1 depois da primeira repetição interna, o continua novamente serve terminar os valores restantes.

3.

A alternativa C está correta. Em UM a declaração variável para i é ilegal. Isto é tipo de declaração só é permitido na primeira parte de um porque () dê laçada. A ausência de inicialização também deveria ser uma pista aqui. Em B o dê laçada controle expressão-o i variável neste case— is de int de tipo. Um vaia que expressão magra é requerida. C é válido. Apesar da complexidade de declarar um valor dentro o porque () construcion, e um fora de (junto com o uso do operador de vírgula na parte de fim) isto é completamente legitime. D teria estado correto, a não ser que a etiqueta foi omitida de linha 2 que deveria ter lido dê laçada: faça.

4.

A alternativa D está correta. O primeiro teste a linha 2 faltas que imediatamente causas controlam para saltar para enfileirar 10 e evitam ambos os possíveis testes que poderiam resultar na produção de mensagem um ou mensagem dois. Assim, embora o teste a linha 3 fosse verdade, nunca é feito; UM não está correto. Em linha 10, o teste está novamente falso, assim a mensagem a linha 11 é saltada, mas mensagem quatro, a linha 14, é produção.

5.

A alternativa D está correta. Um está incorreto porque o código está legal apesar da expressão em linha 5. Isto é porque o próprio expresion é uma constante. B está incorreto porque declara que o interruptor () parte pode levar um argumento longo. Só byte, pequeno, chamusca e int são aceitáveis. A produção é o resultado do valor 2 gosta disto: Primeiro, a opção caso 2: é selecionado, qual valor de produções é dois. Porém, não há nenhuma declaração de fratura entre linhas 4 e 5, assim a execução entra no próximo caso e valor de produções é três de linha 6. A falta: parte de um interruptor () só é executado quando nenhuma outra opção foi selecionada, ou se há nenhuma fratura que precede isto. Neste caso, retifica nenhum destes cabos de situações, assim a produção só consiste nas duas mensagens listadas em D.

6.

As alternativas B, E, e F estão corretas. A exceção causa um salto fora do bloco de prova, assim o Sucesso de mensagem de linha 4 não está impresso. A primeira captura aplicável está em linha 6 que é na partida exata para o excepition lançado. Isto resulta na mensagem a linha 7 ser impresso, assim B é um das respostas exigidas. Só um bloco de captura sempre é executado, assim controla passa o finalmente bloco que resulta na mensagem a linha 16 produção de ser; assim E é parte da resposta correta. Considerando que a exceção foi pegada, é considerado que tem sido dirigido e execução continua depois o finalmente bloco. Isto resulta na produção da mensagem a linha 18, assim F também é parte da resposta correta.

7.

As alternativas A, E, e F estão corretos. Sem exceções o bloco de prova executa a conclusão, assim o Sucesso de mensagem de linha 4 está impresso e UM é parte da resposta correta. Nenhuma captura é executada, assim B, C, e D estão incorretos. Controle então passa o finalmente bloco que resulta na mensagem a linha 16 produção de ser assim E é parte da resposta correta. Porque nenhuma exceção foi lançada, execução continua depois o finalmente bloco, produção de inthe resultante da mensagem a linha 18, assim F também é parte da resposta correta.

8.

A alternativa E está correta. O erro lançado previne conclusão do bloco de prova, assim o Sucesso de mensagem de linha 4 não está impresso. Nenhuma captura é apropriada, assim B, C, e D estão incorretos. Controle então passa o finalmente bloco que resulta na mensagem a linha 16 produção de ser; assim opção E é parte da resposta correta. Porque o erro não foi pegado, execução encerra o método e o erro é rethrown no visitante deste método, assim F não é parte da resposta correta.

9.

A alternativa B está correta. Um daria informação de número de linha enganosa no rastro de pilha do excepition e informa que a exceção surgiu a linha 1 que é onde o objeto de exceção foi criado. C é ilegal desde que você tem que lançar objeto de na que é uma subdivisão de classe de java. Lang. Throwable, e você não pode lançar uma classe, só objeto de na. D também é ilegal, como tenta tothrow um Fio que não é uma subdivisão de classe de java. Lang. Throwable. E é completamente legal, mas não é tão bom quanto B desde que E não leva o esforço para clarificar natureza de thr do problema provendo um fio de explicação.

10.

As alternativas B e D estão corretas. Um não dirige as exceções, assim o aMethod de método poderiam lançar quaisquer das exceções que astucioso () poderia lançar. Porém o excepitions não são declarados com uma construção de lançamentos. Em B que declara "lança IOException" é

suficiente, porque java. Lang. RuntimeExcepition não é um excepition conferido e porque IOExceptionis um superclass de MalformedURLExceptions, é desnecessário mencionar o MalformedURLException explicitamente (embora poderia fazer melhor "ego-documentação" fazer assim). C é inaceitável porque sua declaração de lançamentos não menciona o excepitions—it conferido não é nenhum erro de na para declarar o excepition de runtime, althoughit é estritamente redundante. D também é aceitável, desde que o bloco de captura dirige IOException que inclui MalformedURLException. RuntimeException, ainda será lançado pelo aMethod de método () se é thrownby astucioso (), mas como RuntimeExceptionis não é um excepition conferido, isto é um não erro de na. E não é aceitável, como o método anulando em anotherClass é declarado como lançando IOException, overrddenmethod de whilethe em aClass só foi declarado como lançando MalformedURLEXxception. Teria estado correto para a classe básica declarar que lança IOException e então a classe derivada para lançar MalformedURLException e então a classe derivada para lançar MalformedURLException, mas como é, o método anulando está tentando para lançar exceções não declaradas para o método original. O fato que a única exceção que de fato pode surgir é que o MalformedURLException não é bastante para salvar isto, porque o compilador só confere as declarações, não a semântica do código.

Capítulo 06

1. Considere esta classe:

```
1. public class Test1 {
2. public float aMethod(float a, float b) {
3. }
4. }
```

Qual dos seguintes métodos irá trabalhar de forma legal na linha 4 ?

A. public int aMethod(int a, int b) { }

B. public float aMethod(float a, float b) { }

C. public floata aMethod(float a, float b, int c) thrwos Exception { }

D. public float aMethod(float c, float d) { }

E. private float aMethod(int a, int b, int c) { }

As alternativas A, C, e E estão corretas. Em cada destas respostas, a uma lista de argumento que difere da original, assim o método é sobrecarregado. Métodos sobrecarregados são efetivamente independentes, e não há nenhum constrangimento na acessibilidade, tipo de retorno, ou excepitions que podem ser lançadas. B seria um método anulando legal, a não ser que não pode ser definido na mesma classe como o método original; bastante, deve ser declarado em uma subdivisão de classe. D também é na anulam, como os tipos de seus argumentos é o mesmo: Mudando os nomes de parâmetro não é suficiente a conta como sobrecarregando.

2. Considere estas classes, definidas em arquivos fontes separados:

```
1. public class Test1 {
2.     public float aMethod(float a, float b) throws IOException { }
3. }
```

```
1. public class Test2 extends Test1 { }
```

Qual dos métodos a seguir, trabalha de forma legal na linha 2 da classe Test2 ?

A. float aMethod(float a, float b) { }

B. public int aMethod(int a, int b) throws Exception { }

C. public float aMethod(float a, float b) throws Exception { }

D. public float aMethod(float p, float q) { }

As alternativas B e D estão corretas. Um é ilegal porque é menos acessível que o método original; o fato que é não lança nenhuma exceção é perfeitamente aceitável. B é legalbecause sobrecarrega o método da classe de pai, e como tal não é constrangido por qualquer regra que governa seu valor de retorno, acessibilidade, ou lista de argumento. A exceção lançada por C é suficiente ao ilegal de método de makethat. D é legal porque a acessibilidade e tipo de retorno são idênticos, e o método é na anulam porque os tipos dos argumentos são identical—remember que os nomes dos argumentos são irrelevantes. A ausência de lista de exceção de na em D não é um problema: Na que anula método pode lançar menos exceções que seu original legitimamente, mas pode não lançar mais.

3. Você esteve a favor de um determinado documento de desenho de um sistema de inscrição veterinário para implementação em Java. Declara:

" Um animal tem um dono, uma data de inscrição, e uma data vacinação-devida. Um gato é um animal que indica se foi neutralizado, e uma descrição textual de seu espécie. "

Dado que a classe animal já foi definida, qual dos campos a seguir é apropriado para a inclusão na classe de gato como sócios?

A. Pet thePet;

B. Date registered;

C. Date vaccinationDue;

D. Cat theCat;

E. Boolean neutered;

F. String markings.

As alternativas E e F estão corretas. A classe de Gato é uma subdivisão de classe do Acaricie classe, e como tal deveria estender Acaricia, em lugar de contendo instância de na de Acariciam. B e C deveriam ser os sócios do Acaricie classe e tal é herdado na classe de Gato; então, eles não deveriam ser declarados na classe de Gato. D declararia uma referência a instância de na da classe de Gato que não é geralmente apropriado dentro da própria classe de Gato (a menos que,

talvez, lhe lhe pediram que desse para o Gato um sócio que se refere a sua mãe). Finalmente, o neutered sinalizam e descrições de markings, E e F, são os artigos pedidos pela especificação; estes são artigos corretos.

4. Você esteve a favor de um determinado documento de desenho de um sistema de inscrição veterinário para implementação em Java. Declara:

" Um animal tem um dono, uma data de inscrição, e uma data vacinação-devida. Um gato é um animal que indica se foi neutralizado, e uma descrição textual de seu espécie. "

Dado que a classe animal já foi definida e você espera usar a classe de Cat livremente ao longo da aplicação, como você faria a declaração de abertura da classe Cat, mas não incluindo a primeira cinta de abertura? Use somente estas palavras e espaços: boolean, Cat, Date, extends, Object, Owner, Pet, private, protected, public, String.

Resposta: A classe públic cat estende animal. A classe deveria ser pública desde que será usado livremente ao longo da aplicação. A declaração "Um gato é um animal" nos fala que a classe Cat e sua subdivisão da classe Animal. São requeridas as outras palavras oferecidas para o corpo das definições de Cat ou Pet—for use como variables— mas de sócio não é parte da declaração de abertura.

5. Considere a classe a seguir, ela é declarada em arquivos fontes separados.

```
1.  public class Base {
2.  public void method(int i) {
3.  System.out.println("Value is " + i);
4.  } }
```

```
1.  public class Sub extends Base {
2.  public void method(int j) {
3.  System.out.println("This value is " + j);
4.  }
5.  public void method(String s) {
6.  System.out.println("I was passed " + s);
7.  }
8.
9.  public static void main(String args[ ]) {
10.    Base b1 = new Base( );
11.    Base b2 = new Sub( );
12.   b1.method(5);
13.   b2.method(6);
14. } }
```

Qual o resultado quando o método main da sub classe é executado ?

A. Os valores são 5 e 6

B. Estes valores são 5 e 6

C. O valor é 5 e este valor é 6

D. Este valor é 5 e o valor é 6

E. O valor passado foram 5 e 6.

A alternativa C está correta. A primeira mensagem é produzida pela classe Básica quando b1. Método (5) é chamado e é então Valor é 5. Apesar de variável b2 que é declarada como sendo da classe básica, o comportamento que resulta quando método () é invocado nisto é o comportamento associado com classe do objeto atual, não Com o tipo da variável. Desde que o objeto é de classe Sub, não de classBase, a segunda mensagem é gerada por linha 3 de classe Sub: Este valor é 6.

6. Considere a seguinte definição de classe.

```
1. public class Test extends Base {
2.     public Test(int j) { }
3.
4.     public Test(int j, int k) {
5.     super(j, k);
6. }
7. }
```

Qual das seguintes chamadas é verdadeira ?

A. Test t = new Test( );

B. Test t = new Test(1);

C. Test t = new Test(1, 2);

D. Test t = new Test(1, 2, 3);

E. Test t = (new Base( )). new Test(1);

As alternativas B e C estão corretas. Considerando que a classe tem constructors explícito definido, o constructor de falta é supressed, assim UM não é possível. B e C têm listas de argumento que maych o constructors definiram respectivamente a linhas 2 e 4., e assim é construções corretas. D tem três argumentos de inteiro, mas três são nenhum constructors que levam três argumentos de qualquer tipo na

classe de Teste, assim D está incorreto. Finalmente, E é uma sintaxe usada para construção de classes internas e está então errado.

7. Considere a seguinte definição de classe:

```
1. public class Test extends Base {
2.     public Test(int j) {
3. }
4. public Test(int j, int k) {
5.      super(j, k);
6. } }
```

Qual das seguintes formas de construção devem existir explicitamente na definição da classe Base?

A. Base( ) { }

B. Base(int j) { }

C. Base(int j, int k) { }

D. Base(int j, int k, int l) { }

As alternativas A e C estão corretas. No constructor a linhas 2 e 3, não há nenhuma chamada explícita a qualquer um isto () ou super () que meios que o compilador gerará uma chamada para o zero constructor de superclass de argumento, como em ª A chamada explícita para extranumerário () a linha 5 requer que a classe Básica tem que ter um constructor como em C. Isto tem duas conseqüências. Primeiro, C deve ser um do constructors exigido e então um das respostas. Segundo, a classe Básica tem que ter aquele constructor pelo menos definido explicity, assim o constructor de falta não é gerado, mas deve ser somado explicity. Então o constructor de UM também é requerido e deve ser uma resposta correta. Em nenhum ponto na classe de Teste uma chamada está lá a ou um constructor de superclass com um ou lá argumentos, assim B e D não precisam explicitamente exista.

8. Qual das seguintes declarações é verdadeira ?

A. Uma Inner class permite uma declaração private.

B. Uma Inner class permite uma declaração static.

C. Uma Inner class definida em um método sempre deverá ser anônima.

D. Uma Inner class definida em um método pode acessar todos os métodos com variáveis locais.

E. Construção de Inner class permite a requisição de outra classe.

As alternativas A, B, e E estão corretas. Podem ser definidas classes internas com qualquer acessibilidade, tão privado é completamente aceitável e A está correto. Semelhantemente, o modificador estático é permitido em na classe interna que causa isto não ser associado com qualquer instância particular da classe exterior. Isto significa aquele B também está correto. Classes internas definidas em métodos realmente podem ser freqüentemente anonymous—and are—but que isto não é requerido, assim C está errado. D está errado porque não é possível para na classe interna definiu em um método ter acesso as variáveis locais do método, com exceção dessas variáveis que estão marcado como final. Instância de na construindo de uma classe interna estática não precisa de instância de na do objeto incluindo, mas todo o non—static que classes internas requerem tal uma referência deve estar disponível à operação nova. A referência para o objeto incluindo está comumente implícita como isto que é por que não está comumente explícito. Estes pontos fazem para E retificar.

9. Considere as seguintes definições:

```
1.  public class Outer {
2.    public int a = 1;
3.    private int b = 2;
4.    public void method(final int c) {
5.    int d = 3;
6.    public class Inner {
7.       private void iMethod(int e) {
8.
9.  } } } }
```

Qual das seguintes variáveis irá ser referenciada na linha 8 ?

A. a

B. b

C. c

D. d

E. e

As alternativas A, B, C, e E estão corretas. Desde então Interno não é uma classe interna estática, tem uma referência a na que inclui objeto, e as variáveis daquele objeto são acessíveis. Então A e B estão corretas, apesar do fato que b está marcado privado. Variáveis são só acessíveis essas variáveis são final marcado, assim o argumento de método c está

correto, mas o d variável não é .Finally, o parâmetro e é claro que acessível, desde que é um parâmetro ao método que contém linha 8 isto.

10. Qual das seguintes declarações são verdadeiras ?

A. Dado que Inner é uma classe não-estática declarada dentro de uma classe pública que são definidas formas de construtores Exteriores, e apropriadas, uma instância de Inner pode ser construído assim:
(new Outer()).new Inner()

B. Se uma classe interna anônima dentro da classe Outer é definida para implementar a interface ActionListener, pode ser construída assim:
(new Exterior ()).new ActionListener( )

C. Dado que Inner é uma classe não-estática declarada dentro de uma classe pública que são definidas formas de construtores Exteriores e apropriadas, uma instância de Inner pode ser construída em um método estático assim:

new Inner ()

D. Uma instância de classe anônima que implementa a interface MyInterface pode ser construída e pode ser voltada de um método assim:

```
1. return new MyInterface(int x) {
2. int x;
3. public MyInterface(int x) {
4. this.x = x;
5. } }
```

A alternativa A está correta. Construção de um normal (quer dizer, um nomeou e naõ—static) classe interna requer instância de na da classe incluindo. Freqüentemente esta instância de enclossing é provida pelo inpliede esta referência, mas na que referência explícita pode ser usada na frente da operadora nova, como mostrado na alternativa A.

Classes internas anônimas podem estar só instantiated no mesmo ponto que eles são declarados, assim,:

```
retorno ActionListener novo ()
   actionPerformed nulo público (ActionEvent e);
 ;
```

Conseqüentemente B é de fato illegal—it tenta a instantiate a interface ActionListener como se aquela interface se fosse na classe interna dentro Exterior.

C é ilegal desde Interno é um não—static classe interna, e assim requer uma referência a na que inclui instância quando é construído. A forma mostrada suggeststhe implicaram esta referência, mas desde que o método é estático, há nenhum esta referência e a construção é ilegal.

D é illegalsince que tenta usar argumentos para o constructor de na classe interna anônima que implementa interface de na. A pista está na tentativa para definir um constructor a linha 3. Este não seria um constructor para a interface MyInterface para o class—this interno está errado em dois counts.First, interfaces não definem constructors, e segundo nós precisamos de um constructor para nossa classe anônima, não para a interface.

Capítulo 07

1. Qual das declarações no fragmento de código abaixo é verdadeira ?

```
1. class Greebo extends java.util.Vector implements Runnable {
2. public void run(String message) {
3. System.out.println("in run( ) method: " + message);
4. } }
5.
6. class GreeboTest {
7. public static void main(String args[ ]) {
8.         Greebo g = new Greebo( );
9.         Thread t = new Thread(g);
10.        t.start( );
11.} }
```

A. Haverá um erro de compilação, porque a classe Greebo não implementa a interface Runnable.

B. Haverá um erro de compilação na linha 11, porque você não pode passar um parâmetro para o construtor de uma Linha.

C. O código compilará corretamente mas haverá uma exceção na linha 11.

D. O código compilará corretamente mas haverá uma exceção na linha 12.

E. O código compilará corretamente e executará sem qualquer exceção.

A alternativa A está correta. A interface de Runnable define uma corrida do método com tipo de retorno nulo e nenhum parâmetro. O método cedido ao problema tem um parâmetro de String, assim o compilador reclamará que aquela classe que Greebo não define corrida nula () de interface Runable. B está errado, porque você definitivamente pode passar um parâmetro para o constructor de uma linha; o parâmetro se torna o objetivo da linha. C, D, e E são tolice.

2. Qual das declarações a seguir é a maneira correta para a seguinte aplicação ?

```
1. class HiPri extends Thread {
2. HiPri( ) {
3. SetPriority(10);
4. }
5.
6. public void run( ) {
7. System.out.println("Another thread starting up.");
```

```
8.  While (true) { }
9.  }
10.
11. public static void main(String args [ ]) {
12. HiPri hp1 = new HiPri( );
13. HiPri hp2 = new HiPri( );
14. HiPri hp3 = new HiPri( );
15. hp1.start( );
16. hp2.start( );
17. hp3.start( );
18. } }
```

A. Quando a aplicação é executada, o thread hp1 executará; os threads hp2 e hp3 nunca adquirirão a CPU.

B. Quando a aplicação é executada, todas os três threads (hp1, hp2, e hp3) conseguirão executar e haverá particionamento do tempo na CPU.

C. Uma das alternativas, A ou B será a verdadeira e dependerá da plataforma subjacente.

A alternativa C está correta. A alternativa A é verdadeira em uma plataforma de preemptive, B é verdade em uma plataforma tempo-fatiado. O moral é que tal código deve ser evitado, desde que tais resultados são diferentes em plataformas diferentes.

3. Verdade ou Falso. Um thread quer um segundo thread inelegível para execução. Para fazer isto, o primeiro thread pode chamar o método yield() no segundo thread.

Falso. O método yield( ) é estático e sempre provoca a suspensão do thread atual. Neste caso, ironicamente, é o primeiro thread que se renderá.

4. Verdade ou Falso. Um thread quer um segundo thread inelegível para execução. Para fazer isto, o primeiro thread pode chamar o método suspend() no segundo thread.

Verdade. O segundo thread permanecerá no estado Suspendido até que receba uma chamada do método resume( ).

5. Um thread esta executando o método run( ) incluindo as seguintes linhas:

```
1.  try {
```

```
2.       sleep(100);
3.  }    catch (InterruptedException e) {  }
```

Assumindo que o thread não é interrompido, qual das seguintes declarações é a verdadeira ?

A. O código não compilará, porque exceção pode não ser pegada em um thread que esteja sendo executado através do método run( ).

B. Na linha 2, o thread irá parar o funcionamento. A execução retomará em no máximo 100 milissegundo.

C. Na linha 2, o thread irá parar o funcionamento. A execução retomará exatamente em 100 milissegundo.

D. Na linha 2, o thread irá parar funcionamento. A execução retomará algum tempo depois de 100 milissegundo.

A alternativa D é verdadeira. A linha dormirá durante 100 milissegundo (mais ou menos, depende da resolução do JVM que é usado). Então o thread entrará no estado Pronto; não vai actualy corrido até o scheduler permitir que isto ocorra.

6. Um monitor chamado mon tem 10 threads em seu reservatório de espera; tudos estes threads de espera têm a mesma prioridade. Um dos thread é o thr1. Como você pode notificar o thr1 para que ele possa passar do estado de espera para o estado Pronto?

A. Executando o notify(thr1); de dentro código sincronizado de qualquer mon.

B. Executando o mon.notify(thr1); no código sincronizado de qualquer objeto.

C. Executando o thr1.notify( ); no código sincronizado de qualquer objeto.

D. Executando o thr1.notify( ); de qualquer código (sincronizado ou não) de qualquer objeto.

E. Você não pode especificar qual thread será notificado.

A alternativa E está correta. Quando você chama o método notify( ) em um monitor, você não tem controle sobre o thread de espera que será notificado.

7. Qual das declarações da aplicação abaixo é verdadeira ?

```
1.  class TestThread extends Thread {
```

```
2.  public void run( )
3.          System.out.println(“Starting”);
4.          suspend( );
5.          resume( );
6.          System.out.println(“Done”);
7.  }
8.
9.  public static void main(String args[ ]) {
10. TestThread tt = new TestThread( );
11.     Tt.start( );
12. } }
```

A. A compilação falhará na linha 4, porque o método suspend( ) deve ser chamado no código sincronizado.

B. A compilação falhará na linha 5, porque o método resume( ) deve ser chamado no código sincronizado.

C. A compilação terá sucesso. A execução, será a impressão.

D. A compilação terá sucesso. A execução, somente uma linha de produção (Starting) será impressa.

A alternativa D está correta. Embora os método suspend( ) e resume( ) devam ser chamados de um código sincronizado, não há nenhuma regra correspondente para suspend( ) e resume( ). O código correrá até a linha 4 na qual vai ser suspendida. Considerando que suspendeu, nunca poderá executar linha 5. Uma linha suspendida nunca pode se retomar, desde fazer assim tem que estar correndo; uma linha suspendida só pode ser retomada por uma linha diferente.

8. Qual das declarações da aplicação abaixo é verdadeira ?

```
1.  class TestThread2 extend Thread {
2.  public void run( ) {
3.          System.out.println(“Starting”);
4.           yield( );
5.           resume( );
6.           System.out.println(“Done”);
7.  }
8.
9.  public static void main(String args[ ]) {
10.    TestThread2 tt = new TestThread2( );
11.    tt.start( );
12. } }
```

A. A compilação irá falhar na linha 4, porque o método yield( ) deve ser chamado no código sincronizado.

B. A compilação irá falhar na linha 5, porque o método resume( ) deve ser chamado no código sincronizado.

C. O código será bem sucedido. A execução terá como saída uma impressão.

D. O código será bem sucedido. A execução terá como saída apenas uma linha de impressão.

Neste momento E está correta. A única diferença entre este problema e o anterior é a utilização do método yield( ) na linha 4 em lugar do método suspend( ). A linha que esta executando passa ao estado Pronto; logo posteriormente, o scheduler passa isto atrás ao estado Corrente. Executa a linha 5 e então chama o método resume( ). Esta chamada não tem nenhum efeito, porque a linha não foi suspendida.

9. Se você tentar compilar e executar a aplicação listada abaixo, sempre imprimirá a mensagem “In xxx” ?

```
1. class TestThread3 extends Thread {
2. public void run( ) {
3. System.out.println(“Running”);
4. System.out.println(“Done”);
5. }
6.
7. private void xxx( ) {
8. System.out.println(“In xxx”);
9. }
10.
11. public static void main(String args [ ]) {
12. TestThread3 ttt = new TestThread3( );
13. ttt.xxx( );
14. ttt.start( );
15. } }
```

Sim. A chamada para x x x () acontece antes da linha e é registrado com o thread scheduler, assim a pergunta não tem nada que ver com os threads.

10. Verdadeiro ou Falso. Um monitor de Java deve estender um Thread ou implementar Runnable.

Falso. Um monitor é instância de na de qualquer classe que sincronizou código.

Capítulo 08

1. Dado uma String construída e chamada s = new String ("xyzzy "), qual das chamadas listadas abaixo modificam a String?

A. s.append(“aaa”);

B. s.trim( );

C. s.substring(3);

D. s.replace(‘z’, ‘a’);

E. s.concat(s);

Nenhuma das respostas está correta. Strings são imutáveis.

2. Qual das declarações do código abaixo é verdadeira ?

```
1. String s1 = “abc” + “def”;
2. String s2 = newString(S1);
3. If (s1 == s2)
4.    System.out.println(“== succeeded”);
5. if (s1.equals(s2))
6.    System.out.println(“.equals( ) succeeded”);
```

A. As linhas 4 e 6 serão executadas.

B. A linha 4 será executada e a linha 6 não.

C. A linha 6 será executada e a linha 4 não.

D. Nem a linha 4 nem a linha 6 será executada.

A alternativa C está correta. Considerando que a 1ª e 2ª são referências a dois modificadores, falta um teste. Porém, as cordas contidas dentro dos dois objetos Strings são idênticos, assim o iguals( ) passagens de teste.

3. Suponha que você quer escrever uma classe que oferece para métodos estáticos computar funções trigonométricas hiperbólicas. Você decide que a subdivisão da classe java.lang.Math e provê a funcionalidade nova como jogo de métodos estáticos. Qual declaração abaixo é verdadeira sobre esta estratégia?

A. A estratégia trabalha.

B. A estratégia trabalha, fornecendo os novos métodos públicos.

C. A estratégia trabalha, fornecendo os novos métodos privados.

D. A estratégia falha, porque você não enlata uma subdivisão da classe java.lang.Math.

E. A estratégia falha, porque você não pode somar métodos estáticos para uma subdivisão de classe.

A alternativa D está correta. O java. Lang. Classe de matemática é final, assim não pode ser subclassed.

4. Qual das declarações do fragmento de código abaixo é verdadeira ?

```
1. import java.lang.Math;
2. Math myMath = new Math( );
3. System.out.println(“ consine of 0.123 = “ + myMath.cos(0.123));
```

A. A compilação irá falhar na linha 2.

B. A compilação irá falhar na linha 3.

C. A compilação terá êxito, sendo que o import na linha 1 não é necessário. Durante a execução, uma exception irá ocorrer na linha 3.

D. A compilação terá êxito, sendo que o import na linha 1 é necessário. Durante a execução, uma exception irá ocorrer na linha 3.

E. A compilação terá êxito, e não haverá nenhuma exception durante a execução.

A alternativa A está correta. O constructor para a classe de Matemática é privado, assim não pode ser chamado. Os métodos de classe de Matemática são estáticos, assim isto nunca necessário construir instância de na. A importação a linha 1 não é requerida, como todas as classes do java. Lang empacotam é importado automaticamente.

5. Qual das declarações no fragmento de código abaixo é verdadeira ?

```
1. String s = “abcde”;
2. StringBuffer s1 = new StringBuffer(“abcde”);
3. If (s.equals(s1))
4.    s1 = null;
```

5. if (s1.equals(s))
6. s = null;

A. A compilação irá falhar na linha 1, porque o construtor da String deve ser explicito.

B. A compilação irá falhar na linha 3, porque s e s1 são de tipos diferentes.

C. A compilação terá êxito. Durante a execução, uma exception irá ocorrer na linha 3.

D. A compilação terá êxito. Durante a execução, uma exception irá ocorrer na linha 5.

E. A compilação terá êxito. Não ocorrerá nenhuma exception.

A alternativa E está correta. Um está errado porque linha 1 é um modo perfeitamente aceitável para criar um fio, e é realmente mais eficiente que chamando o constructor explicitamente. B está errado porque o argumento para o s igual () método é de Objeto de tipo; assim qualquer referência de objeto ou variável de ordem pode ser passada. As chamadas em linhas 3 e 5 retorno falso sem lançar exceções.

6. O fragmento de código abaixo irá compilar com sucesso ? Neste caso, a linha 02 será executada com sucesso ?

1. if (“Hedgehog”.startsWith(“Hedge”))
2. System.out.println(“Line 2”);

O código compila, e a linha 2 é executada.

7. Verdadeiro ou Falso. No fragmento de código abaixo, após a execução da linha 1, sbuf referencia uma instancia da classe StringBuffer. Após a execução da linha 2, sbuf ainda referencia a mesma instancia.

1. StringBuffer sbuf = new StringBuffer(“abcde”);
2. Sbuf.insert(3, “xyz”);

Verdadeiro. A classe de StringBuffer é mutable. Depois da execução da linha 2, sbuf referencia o mesmo objeto, embora o objeto tenha sido modificado.

8. Verdadeiro ou falso: No fragmento de código abaixo, após a execução da linha 1, sbuf referencia uma instancia da classe StringBuffer. Após a execução da linha 2, sbuf ainda referencia a mesma instancia.

1. StringBuffer sbuf = new StringBuffer(“abcde”);
2. Sbuf.append(“xyz”);

Verdadeiro. Veja resposta 7 acima.

9. Verdadeiro ou Falso: No fragmento de código abaixo, a linha 4 é executada ?

1. String s1 = “xyz”;
2. String s2 = “xyz”;
3. If (s1 == s2)
4. System.out.println(“Line 4”);

Verdadeiro. A linha 1 constrói uma instância nova da String e armazena isto no depósito de String. Na linha 2, _xyz_ é representado já no depósito, assim nenhuma instância nova é construída.

10. Verdadeiro ou falso. No fragmento de código abaixo a linha 4 é executada.

1. String s1 = “xyz”;
2. String s2 = new String(s1);
3. If (s1 == s2)
4. System.out.println(“Line 4”);

Falso. A linha 1 constrói uma instância nova da String e armazena isto no depósito de String. Na linha 2 é construido explicitamente uma outra instância.

Capítulo 09

1. Um campo de texto é construído e então determinado uma cor de primeiro plano de branco e uma 64-point fontel bold serif . O campo de texto é somado então a um applet que tem uma cor de primeiro plano de vermelho, cor de fundo de azul, e 7-point fonte de sansserif claro. Qual declaração abaixo é verdadeira para o campo texto?

A. Cor de primeiro plano é preta, cor de fundo é branca, fonte é 64-point serif bold corajoso.

B. Cor de primeiro plano é vermelha, cor de fundo é azul, fonte é 64-point serif bold corajoso.

C. Cor de primeiro plano é vermelha, cor de fundo é azul, fonte é 7-point serif bold corajoso.

D. Cor de primeiro plano é branca, cor de fundo é azul, fonte é 7-point serif bold corajoso.

E. Cor de primeiro plano é branca, cor de fundo é azul, fonte é 64-point bold serif .

A alternativa E está correta. Considerando que o botão não especifica um fundo, adquire o mesmo fundo como o applet: azul. São fixados a cor de primeiro plano do botão e manancial explicitamente a branco e 64-point serifa corajoso, assim estas colocações entram em vigor em lugar de os valores do applet.

2. Você tem uma caixa de cheque em um painel; o painel está em um applet. O applet não contém nenhum outro componente. Usando o SetFont( ), você permite que ao applet uma fonte 100-point, e você dá para o painel uma fonte 6-point. Qual declaração ou declarações abaixo está correta?

A. A caixa de cheque usa uma fonte 12-point.

B. A caixa de cheque usa uma fonte 6-point.

C. A caixa de cheque usa uma fonte 100-point.

D. A caixa de cheque usa a fonte do applet, porque você não pode fixar uma fonte em um painel.

E. A caixa de cheque usa a fonte do painel, porque você não fixou uma fonte explicitamente para a caixa de cheque.

As alternativas B e E estão corretas. Considerando que você não tem jogo explicitamente um manancial para a caixa de cheque, usa o manancial de seu imediato.

3. Você tem uma caixa de cheque em um painel; o painel está em um applet. O applet não contém nenhum outro componente. Usando o SetFont( ), você dá para o applet uma fonte 100-point. Qual declaração ou declarações abaixo está correta?

A. A caixa de cheque usa uma fonte 12-point.

B. A caixa de cheque usa uma fonte 6-point.

C. A caixa de cheque usa uma fonte 100-point.

D. A caixa de cheque usa a fonte do applet.

E. A caixa de cheque usa a fonte do painel, porque você não fixou uma fonte explicitamente para a caixa de cheque.

As alternativa C, D, e E estão corretas. O painel não adquire seu manancial explicitamente fixado, assim usa o manancial do applet. A caixa de cheque não adquire seu manancial explicitamente fixado, assim usa o manancial do painel que é o manancial do applet.

4. Você quer construir uma área de texto que tem largura de 80 caráteres e altura de 10 caráteres. Que código usa você?

A. new TextArea(80, 10)

B. new TextArea(10, 80)

A alternativa B esta correta. O número de filas vem primeiro, então o número de colunas.

5. Você constrói uma lista chamando new List(10, false). Qual declaração ou declarações abaixo está correta? (Assuma que o layout managers não modifica a lista de qualquer forma.)

A. A lista tem 10 artigos.

B. A lista apóia seleção múltipla.

C. A lista tem 10 artigos visíveis.

D. A lista não apóia seleção múltipla.

E. A lista adquirirá uma barra de rolagem vertical se precisar.

As alternativas C, D, e E estão corretas. O primeiro parâmetro (10) especifica o número de artigos visíveis. O segundo parâmetro (falso) especifica se seleção de mutiple é apoiada. Uma lista sempre adquire uma barra de rolagem vertical se o número de artigos excede o número de artigos visíveis.

6. Um campo de texto tem uma fontel de largura variável. É construído chamando new TextField(iiiii "). O que acontece se você muda os conteúdos do campo de texto para " wwwww "? (Tenha em mente que i é um dos caráteres de narrowest, e w é um dos mais largos.)

A. O campo de texto fica mais largo.

B. O campo de texto se torna narrower.

C. O campo de texto fica com a mesma largura; ver os conteúdos inteiros você terá que enrolar usando o <- e -> chaves.

D. O campo de texto fica com a mesma largura; veja os conteúdos inteiros você terá que enrolar usando a barra de rolagem horizontal do campo de texto.

A alternativa C está correta. Se um campo de texto é muito estreito para exibir seus conteúdos, você precisa enrolar usando as chaves de seta.

7. O qual dos seguintes itens pode conter um menu? (Escolha uma ou mais.)

A. Um separador

B. Uma caixa de cheque

C. Um menu

D. Um botão

E. Um painel

As alternativas A e C estão corretas. Um menu pode conter artigos de um menu, artigos de um menu de conferir-caixa (não confira caixas!), separador, e (sub)menus.

8. O qual dos seguintes itens pode conter uma barra menu? (Escolha uma ou mais.)

A. Um painel

B. Um frame

C. Um applet

D. Uma barra de menu

E. Um menu

A alternativa B está correta. Só um frame pode conter uma barra de menu.

9. Sua aplicação constrói um frame chamando Frame f = Frame new( ); mas quando você corre o código, o frame não aparece na tela. Que código fará o frame aparecer?

A. f.setSize(300, 200);

B. f.setFont(new Font (SansSerif '', Font.BOLD, 24));

C. f.setForeground(Color.white);

D. f.setVisible(true);

E. f.setSize(300, 200); f.setVisible(true);

A alternativa E está correta. Um frame recentemente construída tem zero-por-zero tamanho e não visível. Você tem que chamar ambos o setSize () (ou setBounds ()) e setVisible ().

10. Verdadeiro ou Falso: A classe de CheckboxGroup é uma subdivisão da Classe de Componentes.

Falso. O java. Awt. Classe de CheckboxGroup não é um Tipo de componente.

Capítulo 10

1. Um programa em Java cria uma check box que usa o código listada abaixo. O programa é corrido em duas plataformas diferentes. O qual das declarações do código a seguir são verdadeiras?

   1. Checkbox cb = new Checkbox(“Autosave”);
   2. Font f = new Font(“Courier”, Font.PLAIN, 14);
   3. Cb.setFont(14);

   Resposta.

   A alternativa E está correta, pois não há nenhum modo de garantir que os botões serão do mesmo tamanho em ambas as plataformas

2. Qual é o resultado que o compilador tentará executar na seguinte aplicação ?

```
1.  import java.awt.*;
2.
3.  public class Q2 extends Frame  {
4.  Q2( ) {
5.            SetSize(300,300);
6.            Button b = new Button(“Apply”);
7.            add(b);
8.   }
9.  public static void main(String args[ ]) {
10.    Q2 that = new Q2( );
11.    that.setVisible(true);
12. }
13. }
```

E está correto. A alternativa A está errado porque o constructor é chamado sua própria classe de dentro; a aplicação compilaria até mesmo se o constructor eram privados. B está errado porque a armação tem uma gerente de plano de falta que é uma instância de BorderLayout. Se você soma () um componente para um recipiente que usa um gerente de plano de Borda, e você não especifica uma região como um segundo parâmetro, então o componente é somado a Centro, da mesma maneira que se você tivesse especificado BorderLayout. CENTRE como um segundo parâmetro. (Nota, porém, que provendo o parâmetro explicitamente está programando muito melhor thanrelying de estilo em comportamento de falta.) C está errado porque o botão aparece; leva a armação inteira, como descreveu em E. Resposta D seria verdade se armações usaram as gerentes de plano de Fluxo através de falta.

3. Qual é o resultado que o compilador tentará executar na seguinte aplicação ?

```
1.  import java.awt.*;
2.  public class Q3 extends Frame {
3.  Q3 {
4.          setSize(300,300);
5.          setLayout(new GridLayout(1,2));
6.
7.          Panel p1 = new Panel( );
8.          p1.setLayout(new FlowLayout(FlowLayout.RIGHT));
9.          p1.add(new Button(“Hello”));
10.         add(p1);
11.
12.          Panel p2 = new Panel( );
13.         p2.setLayout(new FlowLayout(FlowLayout.LEFT));
14.         p2.add(new Button(“Goodbye”));
15.         add(p2);
16.    }
17.
18. public static void main(String args[ ])
19. { Q3 that = new Q3( );
20.   that.setVisible(true);
21.}
22.}
```

C é a alternativa correta. A alternativa A está errado porque o falta plano gerente canbe de qualquer recipiente substituíram; isso é o único modo para adquirir coisas feito se o gerente de falta não é o que você quer. B prejudicam porque não há nenhuma restrição contra havinga única fila ou uma única coluna. O que realmente acontece é isto: A armação contém dois panels—p1 ocupa a esquerda inteira a metade da armação e p2 ocupa o direito inteiro meio (porque a armação usa um grid com uma fila e duas colunas). Cada painel usa um gerente de layuot de Fluxo, assim dentro dos painéis todo componente consegue ser seu tamanho preferido. Assim os dois botões há pouco são grandes bastante para cercar as etiquetas deles/delas. Decore com painel p1 usos um gerente de plano de Fluxo direito-alinhando, assim seu único componente é alinhado ao direito distante daquele painel, só partiu da linha de centro vertical. Decore com painel p2 usos um gerente de plano de Fluxo esquerda-alinhando, assim seu único componente é alinhado o distante partiu daquele painel, só rigkt da linha de centro vertical. O dois fim de botões para cima como descreveu em resposta C. D e E estão incorretos porque os botões conseguem ser os tamanhos preferidos deles/delas.

4. Qual é o resultado que o compilador tentará executar na seguinte aplicação ?

```
1.  import java.awt.*;
2.  public class Q4 extends Frame {
```

```
3.  Q4 ( ) {
4.  setSize(300,300);
5.  setLayout(new GridLayout(3,1));
6.
7.  Panel p1 = new Panel( );
8.  p1.setLayout(new BorderLayout( ) );
9.  p1.add(new Button(“Alpha”), BorderLayout.NORTH);
10. add(p1);
11.
12. Panel p2 = new Panel( );
13. p2.setLayout(new BorderLayout( ) );
14. p2.add(new Button(“Beta”), BorderLayout.CENTER);
15. add(p2);
16.
17. Panel p3 = new Panel( );
18. p1.setLayout(new BorderLayout( ) );
19. p1.add(new Button(“Gamma”), BorderLayout.SOUTH);
20. add(p3);
21.
22. public static void main(String args[ ] ) {
23.  Q4 that = new Q4( );
24.  that.setVisible(true);}
25. }
```

B está correto. A armação é disposta em um grid com threerows e uma coluna. Assim cada do três painel p1, p2, e p3 são tão largos quanto a armação e 1/3 como alto. O “Alfa” botão vai a Norte do painel de topo, assim é tão largo quanto o próprio painel (assim tão largo quanto a armação), e consegue ser sua altura preferida. O “Beta” botão vai a Centro do painel mediano, assim ocupa o painel inteiro (desde que lá é nada mais no painel). O “Gama” botão vai em Sul do painel de fundo, assim isto tão largo quanto o próprio painel (assim tão largo quanto a armação), e consegue ser sua altura preferida.

5. Você gostaria de compilar e executar o código seguinte. Depois que o frame aparece na tela, você gostaria que o resize do frame fosse aproximadamente duas vezes sua largura original e aproximadamente duas vezes sua altura original. Qual das declarações do código seguinte está correta?

```
1.  import java.awt.*;
2.
3.  public class Q5 extends Frame {
4.  Q5( ) {
5.          setSize(300,300);
6.          setFont(new Font(“Helvetica”, Font.BOLD, 36));
7.          Button b = new Button(“Abracadabra”);
8.          add(b, BorderLayout.SOUTH);
```

```
9.  }
10.
11. public static void main(String args[ ]) {
12.   Q5 that = new Q5( );
13.   that.setVisible(true);
14.}
15.}
```

D está correto. A alternativa A está errado porque toda armação adquire uma falta Borda plano gerente. Desde que o botão é colocado em Sul, itnis sempre tão largo quanto a armação, e adquire resized quando a armação adquire resized. Sua altura sempre é sua altura preferida. Nota o do thrree respostas plausíveis (C,D, e E), a resposta correta é o mais simples. O ponto desta pergunta é que quando um recipiente adquire resize, seu gerente de plano dispõe todos os componentes novamente.

6. O código a seguir constrói um GUI com um único botão. Qual declaração é verdadeira a respeito do tamanho do botão?

```
1.  import java.awt.*;
2.  public class Q6 extends Frame {
3.  Q6( ) {
4.  setSize(500,500);
5.  setLayout(new FlowLayout( ) );
6.  Button b = new Button(“ Where am I ?);
7.  Panel p1 = new Panel( );
8.  p1.setLayout(new FlowLayout(FlowLayout.LEFT));
9.  Panel p2 = new Panel( );
10. p2.setLayout(new FlowLayout( ));
11. Panel p3 = new Panel( );
12. p3.setLayout(new FlowLayout( ));
13.
14. p1.add(b);
15. p2.add(p1, BorderLayout.NORTH);
16. p3.add(p2);
17. add(p3);
18. }
19. public static void main(String args[ ]) {
20. Q6.that = new Q6( );
21. That.setVisible(true);
22. } }
```

A alternativa A está correto. As únicas linhas de código que assunto é 9, 10, e 16. O botão é somado a um painel que usa um gerente de plano de Fluxo. Então o botão consegue ser seu tamanho preferido.

7. Em uma aplicação que tem um frame que usa um Border layout manager. Por que provavelmente não será uma boa idéia pôr uma barra de rolagem vertical ao Norte do frame?

Com um gerente de plano de borda, qualquer componente a Norte (ou Sul) é tão largo quanto o recipiente e tão alto quanto sua própria altura preferida. Barra de rolagem de Avertical precisa de bastante playin o verticaldirection, mas não precisa ser muito largo. O problema produz uma barra de rolagem que é ambos muito largo e muito pequeno para ser útil, assim a resposta correta é C. Com um gerente de plano de Borda, barras de rolagem verticais estão muito úteis em Leste e Oeste; barras de rolagem horizontais estão muito úteis em Norte e Sul.

8. Qual é o Layout Manager default para um applet ? Para um frame ? Para um Panel ?

O gerente de plano de falta para painéis e applets é Fluxo. A falta para armações é Borda.

9. Verdadeiro ou Falso: Se um frame usa uma Grid Layout Manager e não contém nenhum painel, então todos os componentes dentro do frame são a mesma largura e altura.

Verdadeiro. O Grid plano gerente ignora o tamanho preferido de componentes e faz para todos os componentes o mesmo tamanho. Se a armação contivesse qualquer painel, então seria provável que os componentes dentro desses painéis sejam menor que esses diretamente contidos pelo painel. Porém, a pergunta explicitamente estados que a armação não contém nenhum painel.

10. Verdadeiro ou Falso: Se um Frame usa seu Layout Manager default e não contém nenhum painel, então todos os componentes dentro do frame são da mesma largura e altura.

Falso. O gerente de plano de falta é Borda. Componentes em Norte e Sul serão a mesma largura; componentes em Leste e Oeste serão a mesma altura. Nenhuma outra generalização é possível.

11. Verdadeiro ou Falso: Com um Border Layout Manager, o componente do Centro adquire todo o espaço em cima do que permanece, depois de componentes em Norte e Sul sido considerado.

Falso. Quase, mas não totalmente. O componente a Centro adquire todo o espaço em cima do que permanece, depois dos componentes a norte, que foram considerados Sul, Leste e Oeste.

12. Verdadeiro ou Falso: Com um Grid Layout Manager, é honrada a largura preferida de cada componente, enquanto altura é ditada; se há muitos componentes para ajustar em única fila, são criadas filas adicionais.

Falso. A pergunta descreve um hodgepodge de atributos de gerente de plano.

Capítulo 11

1. Verdadeiro ou Falso: O modelo de evento de delegação, introduzido na versão 1.1 do JDK, é completamente compatível com o modelo de evento da versão 1.0.

Falso. Os dois modelos de evento são incompatíveis, e eles não deveriam aparecer no mesmo programa.

2. Qual declaração ou declarações são verdades sobre o código listado abaixo ?

```
1.  public class MyTextArea extends TextArea {
2.      public MyTextArea( int nrows, int ncols) {
3.          enableEvents(AWTEvent.TEXT_EVENT_MASK);
4.  }
5.
6.  public void processTextEvent(TextEvent te) {
7.     System.out.println(“Processing a text event.”);
8.  }
9.  }
```

As alternativas A, B, e E estão corretas. Considerando que a classe é pública, tem que residir em arquivo cujo nome corresponde ao nome de classe. Se a chamada para extranumerário (nrows, ncols) é omitido, serão invocados o constructor de nenhum-argumentos para TextArea, e o número desejado de filas e colunas será ignorado. C e D são tenta criar confusão introduzindo conceitos do 1.0 modelo; no modelo de delegação, todos os manipuladores de evento têm tipo de retorno nulo. E está correto porque se a linha sugerida é omitida, willbe de ouvintes de texto ignoraram.

3. Qual declaração ou declarações são verdades sobre o código listado abaixo?

```
1.  public class MyFrame extends Frame {
2.     public MyFrame(String title) {
3.         super(title);
4.            enableEvents(AWTEvent.WINDOW_EVENT_MASK);
5.  }
6.
7.  public void processWindowEvent(WindowEvent we) {
8.        System.out.println(“Processing a window event.”);
9.    }
10.  }
```

As alternativas C e D estão corretas. O código compilará e executará completamente. Porém, sem uma chamada para extranumerário.

ProcessWindowEvent (Nós), o componente não notificará seus ouvintes de janela.

4. Qual declaração ou declarações são verdades sobre o código listado abaixo? (Assuma que as classes F1 e F2 implementam a interface FocusListener.)

```
1. TextField tf = new TextField(“ Not a trick question”);
2. FocusListener flis1 = new F1( );
3. FocusListener flis2 = new F2( );
4. tf.addFocusListener(flis1);
5. tf.addFocusListener(flis2);
```

A alternativa C está correta. Linhas 2 e 3 constroem instâncias das classes de ouvinte, e referências de loja para essas instâncias em variáveis com tipos de interface; tal tarefa é perfeitamente legal. A implicação de resposta que B é que somando um segundo listenermight criam um problema, a delegação suports modelo os ouvintes múltiplos. O código compila completamente e corre sem lançar excepition de na.

5. Qual declaração ou declarações são verdades sobre o código listado abaixo? (Assuma que as classes F1 e F2 implementam a interface FocusListener.)

```
1. TextField tf = new TextField(“Not a trick question”);
2. FocusListener flis1 = new F1( );
3. FocusListener flis2 = new F2( );
4. tf.addFocusListener(flis1);
5. tf.addFocusListener(flis2);
6. tf.removeFocusListener(flis1);
```

A alternativa E está correta. Este problema há pouco está como o prévio, com a adição de um istener de removeFocusL perfeitamente legal () chamada.

6. Qual declaração ou declarações são verdadeiras sobre o código listado abaixo ?

```
1. class MyListener extends MouseAdapter implements MouseListener {
2. public void mouseEntered(MouseEvent mev) {
3.       System.out.println(“Mouse entered.”);
4. }
5. }
```

As alternativas A e C estão corretas. Desde que a classe estende MouseAdapter, e MouseAdapter implementa o istener de MouseL conectam, o istener de MyL classificam implicitly implementa bem a

interface como; não faz nenhum dano para declarar o implementação explicitamente. A classe pode servir como uma ouvinte de rato. Em resposta para eventos de rato diferente de rato entrada, o ouvinte executa os métodos de manipulador que herda de seu superclass; estes métodos não fazem nada.

7. Qual declaração ou declarações são verdades sobre o código listado abaixo?

Sugestão: As cada uma das interfaces ActionListener e ItemListener define um único método.

```
1. class MyListener implements ActionListener, ItemListener {
2. public void actionPerformed(ActionEvent ae) {
3.     System.out.println("Action");
4. }
5. public void itemStateChanged(ItemEvent ie) {
6.     System.out.println("Item");
7. }
8. }
```

A alternativa A está correta. Implementação de interface múltiplo é legal em Java. O classmust implementam todos os métodos de ambas as interfaces, e este realmente é o caso. Desde os utensílios de classe conectam o istener de ActionL, é um ouvinte de ação legal; desde isto também implementa o istener de ItemL conectam, também é um ouvinte de artigo legal.

8. Qual declaração ou declarações são verdades sobre o código listado abaixo?

```
1. class MyListener extends MouseAdapter, KeyAdapter {
2. public void mouseClicked(MouseEvent mev) {
3.     System.out.println("Mouse clicked.");
4. }
5. public void keyPressed(keyEventEvent kev) {
6.    System.out.println("KeyPressed");
7. }
8. }
```

A alternativa A está correta. Esta classe tenta herança de classe múltipla que é ilegal em Java.

9. Verdadeiro ou Falso: Uma sub-classe de componente que executou enableEvents () habilita o processamento de um certo tipo de evento que não pode usar um adaptador como um ouvinte para o mesmo tipo de evento.

Falso. Um componente, se ou não chamou enableEvents explicitamente (), pode ter um unlimitednumber de ouvintes, e essas ouvintes podem ser subdivisões de classe de adaptador.

10. Assuma que a classe AcLis implementa as interfaces de ActionListener. O fragmento de código debaixo que constrói um botão e dá isto a quatro ouvintes de ação. Quando o botão é apertado, qual ouvinte de ação é o primeiro a adquirir seu actionPerformed () que o método invocou?

```
1. Button btn = new Button(“Hello”);
2. AcLis a1 = new AcLis( );
3. AcLis a2 = new AcLis( );
4. AcLis a3 = new AcLis( );
5. AcLis a4 = new AcLis( );
6. btn.addActionListener(a1);
7. btn.addActionListener(a2);
8. btn.addActionListener(a3);
9. btn.addActionListener(a4);
10. btn.removeActionListener(a2);
11. btn.removeActionListener(a3);
12. btn.addActionListener(a3);
13. btn.addActionListener(a2);
```

A alternativa E está correta. Não há nenhuma garantia sobre a ordem de prece de ouvintes de evento.

Capítulo 12

1. Como você fixaria a cor de um contexto de gráficos chamado g a cyan?

A alternativa A está correto. As 13 cores pre-definidas são varibles estático em Cor de classe, assim você os tem acesso pelo nome de classe como você os tem acesso nome de classe como você iria qualquer outra variável estática. O nome do método de cor-colocação é setColor (), não setCurrentColor ().

2. O código das linhas abaixo faz um desenho. Que cor é a linha?

1. g.setColor(Color.red.green.yellow.red.cyan);
2. g.drawLine(0,0,100,100);

A alternativa D (cyan) está correta. Esta pergunta testa seu conhecimento de variáveis estáticas como wellas a classe de Cor. A classe de Cor tem 13 variáveis estáticas finais, nomeado vermelho, verde, amarele, e assim por diante. Este varibles acontecem para ser de Cor de tipo. Assim Cor. Vermelho é o nome de instância de na de Cor. Recorde de Capítulo 3 (modificadores) que há dois modos para ter acesso uma variável estática: pelo nome de classe que é o modo preferido ou por uma referência para qualquer instância da classe. Assim um (non-preferiu) modo para ter acesso a variável estática verde está por Cor. Vermelho, porque Cor. vermelho é uma referência a instância de na. Assim Cor. Vermelho. verde é um modo legal para se referir à variável estática verde. Semelhantemente, o modo preferido para se referir à variável estática amarela é Cor. Amarele, mas é legal (embora muito estranho) para referência isto como cor. Vermelho. Verde. Amarele, porque Cor. Vermelho. Verde é uma referência a instância de na. E assim por diante. A resposta ainda seria cyan se a cor fosse fixada para Colorir seja fixado para Colorir. Vermelho. Bocado. Vermelho. Negro. Cyan. Magenta. Azul. Rosa. Laranja. Cyan.

3. Que desenho o código seguinte faz?

1. g.setColor(Color.black);
2. g.drawLine(10,10,10,50);
3. g.setColor(Color.red);
4. g.drawRect(100,100,150,150);

A alternativa B (uma linha vertical preta que é 40 long,and de pixels uma praça vermelha com lados de 150 pixels). O setColor () método só afeta subseqüentemente gráficos tirados; não afeta gráficos previamente tirados. Assim a linha é preta e a praça é vermelha. Os argumentos para setLine () é coordena de fim-pontos, assim a linha vai de (10,10) para (10,50) e sua duração é 40 argumentos de

pixels.The a drawRect () é x posicionam, posição de y, largura, e altura, assim o lado da praça é 150 pixels.

Alguns leitores podem sentir que uma resposta diferente é apropriada: "Nenhum do anterior, Becauseyou nunca disse que g era Gráficos de instanceof de na." Isto é legítimo; o real assunto é o que fazer quando você tem esta reação durante o Exame de Certificação. Sempre tem em mente que as perguntas de exame estão sobre Java, não sobre retórica. O exame testa seu Conhecimento de Java, não sua habilidade para ver por phrasing enganador.

4. A figure abaixo mostra duas formas. Qual forma (A ou B) id é desenhado pela linha do código seguinte ?

g.fillArc(10,10,100,100,0,90);

A alternativa A está correta. O fillArc () método puxa pedaços de torta, não cordas.

5. Qual das declarações abaixo é verdade? (Escolher um ou mais)

As alternativas B, D, e E estão corretas. Um polyline nunca está cheio ou fechado; é só openrun de na de segmentos de linha. Um polígono pode ser enchido (o fillPolygon () method)or não encheram (o drawPolygon () método).

6. Verdadeiro ou Falso: Quando o thread GUI chama o método paint() para consertar dano de exposição, o método paint() tem que determinar o que foi danificado e foi fixado na sua região de clipe adequadamente.

Falso. Quando há dano para ser consertado, o GUI enfiam passa para pintar () um contexto de gráficos cuja região de clipe já é fixada à região danificada. Java foi construído para ter certeza deste modo aquele programmersnever têm que determinar regiões de clipe danificadas. De fato, os programadores nunca têm que fazer qualquer coisa a em toda parte dano de exposição, contanto todo o desenho é acabado em pintura () ou em métodos chamados por pintura ().

7. O seu manipulador de evento mouseDragged () e seu método paint () fazem assim:

```
1. public void mouseDragged(MouseEvent e) {
2. mouseX = e.getX( );
3. mouseY = e.getY( );
4. repaint( );
5. }
6. public void paint(Graphics g) {
7.    g.setColor(Color.cyan);
8.    g.drawLine(mouseX, mouseY, mouseX+10, mouseY+10);
9. }
```

Você quer modificar seu código de forma que as linhas de cyan acumule na tela, em lugar de ter sido apagado toda vez repaint () chame o método update(). Qual é o modo mais simples para proceder?

A alternativa D está correta, e é uma técnica standard sempre que você não quer atualize () enxugar a tela antes de chamar pintura (). Todas as linhas de cyan diagonais permanecerão na tela; o efeito estará como desenho com uma caneta de caligraphy. Respostas UM e B (em linha 4, substitua repinte () com pintura () ou repinta ()) não compilará, porque ambos pintura () e repinta () requeira uns Gráficos como na introduza. Resposta C é dificuldade séria: super. Atualize (g) clareará a tela e chamará pintura (g) que chamará super. Atualize (g), e assim por diante sempre.

8. Que código você usaria para construir uma font de 24-point do tipo serif ?

A alternativa D está correta. A assinatura para o constructor de Manancial é Manancial (fontname de fio, int nomeiam, int classificam segundo o tamanho). O nome de manancial pode ser um de "Serifa", "SansSerif", ou "Monospaced." O estilo deveria ser um de Manancial. PLANÍCIE, manancial. TIPO NEGRITO, ou Manancial. ITÁLICO.

9. O que faz o método de desenho paint( ) ?

```
1. public void paint(Graphics g) {
2.   g.drawString("question  #9", 10,0);
3. }
```

A alternativa B está correta. O parâmetro de y-coordenada passou em drawString () é a posição vertical do baseline do texto. Desde que o baseline está nas 0 (quer dizer, o topo do componente) só descenders serão visíveis. O fio "pergunta #9 "contém um descender, tão só uma única curva pequena descendente do q será vista.

10. O que faz o método de desenho paint( ) ?

```
1.  public void paint(Graphics g) {
2.     g.drawOval(100,100,44);
3.  }
```

A alternativa D está correta. A assinatura para drawOval () é drawOval (int x, int y, largura de int, altura de int), Onde x e y definem que o canto de superior-esquerda do oval está saltando caixa, e largura e altura definem o tamanho da caixa saltando. A pergunta mostra misconception comum que x e y definem o centro do oval.

Capítulo 13

1. Qual linha ou linhas do código HTML seguinte são errôneas?

```
1. <APPLET WIDTH=50 HEIGHT=95 CODE=Thermostat.Class>
2. <PARAM NAME=scale VALUE=Celsius>
3. </APPLET>
```

Há um problema na linha 1. O valor de CÓDIGO é caso-sensível e capitaliza a primeira carta assim do. Extensão de classe não é válida. O browser ignorarão a etiqueta de applet inteira.

2. Verdadeiro de Falso: O valor de CÓDIGO em uma etiqueta <APPLET> tem que nomear um arquivo de classe que está no mesmo diretório como a chamada página de HTML.

Falso. Se o valor é um URL, então o arquivo de classe poderia residir até mesmo em uma máquina diferente.

3. Verdadeiro ou Falso: Se o returns nulo do getParameter (), assinalando o valor de retorno de uma variável do tipo String podem causar uma exceção a ser lançada.

Falso. Uma variável do tipo String pode ser nomeada com um valor de nulo.

4. Verdadeiro ou Falso: Toda etiqueta de ARQUIVO especifica um arquivo de JARRO exatamente.

Falso. Na ARCHIVE etiqueta pode especificar uma lista vírgula-separada de arquivos de Jarro.

5. Considere o seguinte método init() de um applet:

```
1. public void init( ) {
2.  String val = “Primo”;
3. val = getParameter(“XX”).toUpperCase( );
4. System.out.println(“val = “ + val);
5. }
```

Na linha 4, qual é o valor de val se a etiqueta do applet fosse a seguinte:

```
1. <APPLET CODE=Q5.class WIDTH=100 HEIGHT=100>
2. <PARAM NAME=Xx VALUE=Secondo>
3. </APPLET>
```

A alternativa E está correta. A chamada para getParameter () ignora o caso seu argumento e lucros “secondo” que é o valor exato da

página de HTML Com capitalização intato. A chamada para toUpperCase () converte a caso superior.

6. Considere o seguinte método init() de um applet:

```
1. public void init( ) {
2.  int i =100;
3. String val = getParameter("YY");
4. try {
5.    i = Integer.parseInt(val);
6. } catch (Exception e) { }
7. System.out.println("i = " + i);
8. }
```

Na linha 7, qual é o valor de i se a etiqueta do applet fosse a seguinte:

```
1. <APPLET CODE=Q6.class WIDTH=100 HEIGHT=100>
2. <PARAM NAME=yy VALUE=3a6>
3. </APPLET>
```

A alternativa B está correta. A chamada para getParameter () ignora o caso de seu argumento, e lucros "3 uns 6" que é o valor exato da página de HTML com capitalização intato. O parseInt () chame lança uma exceção de número-formato, assim a tarefa para i em linha 5 nunca acontece. Assim i retém seu valor original de 100.

7. Verdadeiro ou Falso: Na etiqueta seguinte, WhereAmI.classe deve ser achada em big.jar.

```
1. <APPLET CODE=WhereAmI.class HEIGHT=200 WIDTH=75
   ARCHIVE=big.jar>
2. </APPLET>
```

Falso. O arquivo de classe poderia residir em grande. Chocalhe, ou poderia residir no mesmo diretório como o arquivo de HTML.

8. O HTML abaixo é parte da página de Rede localizada em http://www.barra.com. O HTML trabalhará ?

```
1. <APPLET CODE=bodies.Globe.class WIDTH=75 HEIGHT=75
2. CODEBASE=http://www.foo.com/astronomy>
3. </APPLET>
```

Sim. O valor de CODEBASE pode ser na URL.

9. Suponha você tem a linha seguinte etiqueta dentro um <APPLET> :

<PARAM NAME=gem VALUE=pearl>

e a linha seguinte dentro do método init( ) corresponde:

String gem = getParameter(“pearl”0;

Que valor é assinalado para gem ?

O valor é nulo. O getParameter () chamada está procurando um parâmetro nomeado “pérola”, e nenhum tal parâmetro é definido no código de HTML. O único parâmetro definido no HTML é “pedra preciosa.”

10. Considere o método init( ) seguinte:

```
1. public void init( ) {
2. init i = 100;
3. String val = getParameter(“YY”);
4. try  { i = Integer.parseInt(val); }
5. catch (NumberFormatException e) { i = 200;}
6. catch (NullPointerException e) { i = 300;}
7. System.out.println(“ i = 1 “ + i);
8. }
```

Na linha 8, qual é o valor de i se a etiqueta do applet fosse a seguinte:

```
1. <APPLET CODE=Q6.class WIDTH=100 HEIGHT=100>
2. <PARAM NAME=aa VALUE=3a6>
3. </APPLET>
```

A alternativa D está correta. Desde que o aa de parâmetro não é definido, nulo é passado no parseInt () chame a linha 5 e resulta em uma exceção de nulo-ponteiro. O manipulador de exceção de nulo-ponteiro nomeia um valor de 300 a i.

Capítulo 14

1. Qual das declarações abaixo é verdadeira?

Só D está correto. Caráter de UTF são tão grande quanto eles precisam ser. Caráter de Unicode são todos os 16 pedaços. Não há nenhuma tal coisa como um caráter de Bytecode; bytecode é o formato gerado pelo Java compile.

2. Qual das declarações abaixo é verdadeira?

Todas as três declarações são falsas. Construção e coleção de lixo de um arquivo não têm nenhum efeito no sistema de arquivo local.

3. Verdadeiro ou Falso: A classe de Arquivo contém um método que muda o diretório de funcionamento atual:

Falso A classe de Arquivo não provê um modo para mudar o diretório de funcionamento atual

4. Verdadeiro de Falso: É possível usar a classe de Arquivo para listar os conteúdos do diretório de funcionamento atual.

Verdadeiro. O código abaixo de espetáculos como isto é acabado:

Arquive f = Arquivo novo (“.”)

String conteúdos [] = f. lista ();

5. Quantos bytes escreve o código seguinte para arquivar destfile?

```
1. try {
2. FileOutputStream fos = new FileOutputStream(“destfile”);
3. DataOutputStream dos = new DataOutputStream(fos);
4. dos.writeInt(3);
5. dos.writeDouble(0.0001);
6. dos.close( );
7. fos.close( );
8. }
9. catch (IOException e ) {  }
```

A alternativa C está correta. O writeInt () chamada escreve int de na que é 4 bytes longo; o writeDouble () chamada escreve fora um dobro que é 8 bytes longo para um total de 12 bytes.

6. O que imprime o fragmento de código seguinte a linha 9?

```
1. FileOutputStream fos = new FileOutputStream(“xx”);
```

```
2. for (byte b=10; b<50; b++)
3.    fos.write(b);
4. fos.close();
5. RandomAccessFile raf = new RandomAccessFile("xx", "r");
6. Raf.seek(10);
7. int i = raf.read( );
8. raf.close( );
9. System.out.println("i = " + i);
```

A alternativa B está correta. Todo o código é perfeitamente legal, assim nenhum excepitions são lançados. O primeiro byte no arquivo é 10, os próximos bytes são 11, o próximo é 12, e assim por diante. Os bytes a arquivo posição 10 é 20, assim a produção é i = 20.

7. Um arquivo é criado com o código seguinte:

```
1. FileOutputStream fos = new FileOutputSteam("datafile");
2. DataOutputStream dos = new DataOutputStream(fos);
3. For (int i =0; i < 500; i++)
4.    dos.writeInt(i);
```

Você gostaria de escrever código para ler os dados atrás deste arquivo. Quais soluções listadas abaixo trabalharão?

As alternativas A e D estão corretos. Solução UMAS cadeias um dados introduz fluxo sobre um arquivo introduza fluxo. Solução D usa a classe de RandomAccessFile simplesmente. B falha porque a classe de FileReader não tem nenhum readInt () método; os leitores e escritores só dirigem texto. Solução que C falha porque a classe de PipedInputStream não tem nada que ver com arquivo eu / º (Transportou introduza e fluxos de produção são usados em comunicação de interthread.) Solução que E falha porque você não pode encadear que um dados introduz fluxo sobre um leitor de arquivo. Leitores leram serviços domésticos, e introduz bytes de manivela de fluxos.

8. Verdadeiro ou Falso: Leitores têm métodos que podem ler retorno flutua e dobra.

Falso. Os leitores e escritores só lidam com caráter eu / º

9. Você executa o código abaixo em um diretório vazio. O que é o resultado?

```
1. File f1 = new File("dirname");
2. File f2 = new File(f1,"filename");
```

A alternativa E está correta. Instância de na construindo da classe de Arquivo não tem nenhum efeito no sistema de arquivo local.

10. Qual é o resultado tentando compilar e executar no fragmento de código abaixo? Assuma que o fragmento de código é parte de uma aplicação que tem escreve permissão no diretório de funcionamento atual. Também assuma que antes de execução, não contém o diretório de funcionamento atual que um arquivo chamado datafile.

```
1. try {
2. RandomAccessFile raf = new RandomAccessFile("datafile"), "rw";
3. BufferedOutputStream bos = new BufferedOutputStream(raf);
4. DataOutputStream dos = new DataOutputStream(bos);
5. dos.writeDouble(Math.PI);
6. dos.close( );
7. bos.close( );
8. raf.close( );
9. } catch (IOException e) { }
```

A alternativa A está correta. Compilação falha a linha 3, porque não há nenhum constructor para BufferedOutputStream que leva um objeto de RandomAccessFile como um parâmetro. Você pode estar seguro deste até mesmo se você não está familiarizado com produção de buffered que flui, porque arquivos de fortuito-acesso são completamente incompletely incompatível com o fluxo / reader/writer modelo.