N

Utilizando o depurador

Fomos criados para cometer equívocos, programados para erro.
—Lewis Thomas

O que antecipamos raramente ocorre; o que menos esperamos geralmente acontece.
—Bejamin Disraeli

Ele pode correr, mas não pode se esconder.
—Joe Louis

Uma coisa é mostrar a um homem que ele está errado e outra, é colocá-lo de posse da verdade.
—John Locke

OBJETIVOS

Neste capítulo, você aprenderá:

- Configurar pontos de interrupção para depurar aplicativos.
- Utilizar o comando `run` para executar um aplicativo por meio do depurador.
- Utilizar o comando `stop` para configurar um ponto de interrupção.
- Utilizar o comando `cont` para continuar a execução.
- Utilizar o comando `print` para avaliar expressões.
- Utilizar o comando `set` para alterar valores de variáveis durante a execução do programa.
- Utilizar os comandos `step`, `step up` e `next` para controlar a execução.
- Utilizar o comando `watch` para ver como um campo é modificado durante a execução do programa.
- Utilizar o comando `clear` para listar pontos de interrupção ou remover um ponto de interrupção.

Sumário

N.1 Introdução

No Capítulo 2, você aprendeu que há dois tipos de erros — erros de sintaxe e erros de lógica — e aprendeu a eliminar erros de sintaxe do seu código. Erros de lógica não impedem que o aplicativo compile com sucesso, mas fazem com que um ele produza resultados errôneos quando executado. O JDK 5.0 inclui um software chamado de **depurador** que permite monitorar a execução dos seus aplicativos para que você possa localizar e remover erros de lógica. O depurador será uma das suas ferramentas mais importantes de desenvolvimento de aplicativos. Muitos IDEs fornecem depuradores próprios semelhantes àquele incluído no JDK ou fornecem uma interface gráfica com o usuário para o depurador do JDK.

Este apêndice demonstra os recursos-chave do depurador do JDK que utilizam aplicativos de linha de comando e não recebem nenhuma entrada do usuário. Os mesmos recursos de depurador discutidos aqui podem ser usados para depurar aplicativos que recebem a entrada do usuário, mas depurar esses aplicativos requer uma configuração um pouco mais complexa. Para focalizarmos os recursos do depurador, optamos por demonstrá-lo com aplicativos de linha de comando simples que não envolvem nenhuma entrada de usuário. Fornecemos instruções para depurar outros tipos de aplicativos no nosso site da Web em `www.deitel.com/books/jhtp6/index.html`. Você também pode encontrar informações adicionais sobre o depurador Java em `java.sun.com/j2se/5.0/docs/tooldocs/windows/ jdb.html`.

N.2 Pontos de interrupção e os comandos `run`, `stop`, `cont` e `print`

Iniciamos nosso estudo sobre o depurador investigando os **pontos de interrupção**, marcadores que podem ser configurados em qualquer linha executável do código. Quando a execução do aplicativo alcança um ponto de interrupção, a execução efetua uma pausa, permitindo que você examine os valores das variáveis para ajudar a determinar se há erros de lógica. Por exemplo, você pode examinar o valor de uma variável que armazena o resultado de um cálculo a fim de determinar se o cálculo foi realizado corretamente. Observe que configurar um ponto de interrupção em uma linha de código que não seja executável (como um comentário) faz com que o depurador exiba uma mensagem de erro.

Para ilustrarmos os recursos do depurador, utilizaremos o aplicativo `AccountTest` (Figura N.1) que cria e manipula um objeto da classe `Account` (Figura 3.13). A execução de `AccountTest` inicia em `main` (linhas 7–24). A linha 9 cria um objeto `Account` com um saldo inicial de US$ 50.00. Lembre-se de que o construtor de `Account` aceita um argumento, que especifica o `balance` (saldo) inicial de `Account`. As linhas 12–13 geram a saída do saldo inicial na conta por meio do método `getBalance` de `Account`. A linha 15 declara e inicializa uma variável local `depositAmount`. As linhas 17–19 então imprimem `depositAmount` e o adicionam ao `balance` da `Account` utilizando seu método `credit`. Por fim, as linhas 22 e 23 exibem o novo `balance`. [*Nota*: O diretório de exemplos do Apêndice N contém uma cópia do `Account.java` idêntica àquela da Figura 3.13.]

Nas etapas a seguir, você utilizará pontos de interrupção e vários comandos de depurador para examinar o valor da variável `depositAmount` declarada em `AccountTest` (Figura N.1)

1. *Abrindo a janela Prompt do MS-DOS e mudando de diretório.* Abra a janela **Prompt do MS-DOS** selecionando **Iniciar > Programas > Acessórios > Prompt do MS-DOS**. Mude para o diretório que contém os exemplos do apêndice digitando `cd C:\examples\debugger` [*Nota:* Se seus exemplos estiverem em um diretório diferente, utilize esse diretório aqui.]
2. *Compilando o aplicativo para depuração.* O depurador Java só funciona com os arquivos `.class` compilados com a opção de compilador **-g**, que gera informações utilizadas pelo depurador para ajudar a depurar seus aplicativos. Compile o aplicativo com a opção de linha de comando `-g` digitando `javac -g AccountTest.java Account.java`. Lembre-se, no Capítulo 2, de que esse comando compila tanto `AccountTest.java` como `Account.java`. O comando `java -g *.java` compila todos os arquivos no `.java` do diretório funcional para depuração.

```
1  // Fig. N.1: AccountTest.Java
2  // Cria e manipula um objeto Account.
3
4  public class AccountTest
5  {
```

Figura N.1 A classe `AccountTest` cria e manipula um objeto `Account`. (Parte 1 de 2.)

```
6     // método main inicia a execução
7     public static void main( String args[] )
8     {
9        Account account = new Account( 50.00 ); // cria o objeto Account
10
11       // exibe o saldo inicial do objeto Account
12       System.out.printf( "initial account balance: $%.2f\n",
13          account.getBalance() );
14
15       double depositAmount = 25.0; // deposita uma quantia
16
17       System.out.printf( "\nadding %.2f to account balance\n\n",
18          depositAmount );
19       account.credit( depositAmount ); // adiciona ao saldo da conta
20
21       // exibe um novo saldo
22       System.out.printf( "new account balance: $%.2f\n",
23          account.getBalance() );
24    } // fim de main
25
26 } // fim da classe AccountTest
```

```
initial account balance: $50.00

adding 25.00 to account balance

new account balance: $75.00
```

Figura N.1 A classe `AccountTest` cria e manipula um objeto `Account`. (Parte 2 de 2.)

3. *Iniciando o depurador.* No **Prompt do MS-DOS**, digite `jdb` (Figura N.2). Esse comando iniciará o depurador Java e permitirá que você utilize os recursos dele. [*Nota*: Modificamos as cores da nossa janela **Prompt do MS-DOS** para destacar em amarelo a entrada de usuário requerida por cada passo.]
4. *Executando um aplicativo no depurador.* Execute o aplicativo `AccountTest` por meio do depurador digitando `run AccountTest` (Figura N.3). Se você não configurar pontos de interrupção antes de executar seu aplicativo no depurador, o aplicativo executará como se estivesse utilizando o comando `java`.
5. *Reiniciando o depurador.* Para utilização adequada do depurador, você deve configurar pelo menos um ponto de interrupção antes de executar o aplicativo. Reinicie o depurador digitando `jdb`.
6. *Inserindo pontos de interrupção em Java.* Você configura um ponto de interrupção em uma linha específica do código no seu aplicativo. Os números das linhas usados nestes passos são provenientes do código-fonte da Figura N.1. Configure um ponto de interrupção na linha 12 no código-fonte digitando `stop at AccountTest:12` (Figura N.4). O **comando `stop`** insere um ponto de interrupção no número da linha especificada depois do comando. Você pode configurar quantos pontos de interrupção forem necessários. Configure um outro ponto de interrupção na linha 19 digitando `stop at AccountTest:19` (Figura N.4). Quando o aplicativo executa, ele interrompe a execução em qualquer linha que contenha um ponto de interrupção. Diz-se que o aplicativo está no **modo de interrupção** (*break mode*) quando o depurador efetua uma pausa na execução do aplicativo. Pontos de interrupção podem ser até mesmo configurados depois de o processo de depuração ter iniciado. Observe que o comando de depurador `stop in`, seguido por um nome de classe, um ponto e um nome de método (por exemplo, `stop in Account.credit`) instrui o depurador a configurar um ponto de interrupção na primeira instrução executável no método especificado. O depurador efetua uma pausa na execução quando o controle do programa entra no método.

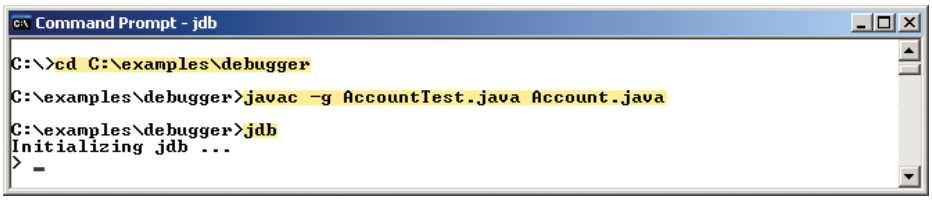


Figura N.2 Iniciando o depurador Java.

Figura N.3 Executando o aplicativo `AccountTest` por meio do depurador.

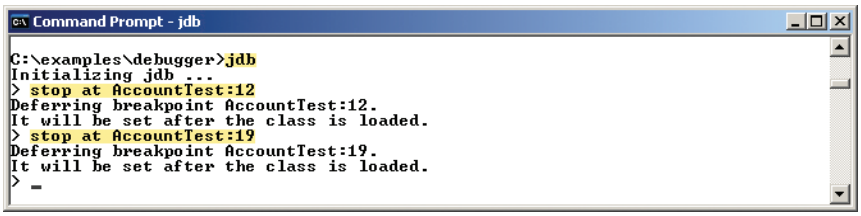


Figura N.4 Configurando pontos de interrupção nas linhas 12 e 19.

7. *Executando o aplicativo e começando o processo de depuração.* Digite `run AccountTest` para executar seu aplicativo e começar o processo de depuração (Figura N.5). Observe que o depurador imprime um texto indicando que os pontos de interrupção foram configurados nas linhas 12 e 19. O depurador identifica cada ponto de interrupção como um 'deferred breakpoint' (ponto de interrupção adiado), pois cada um foi configurado antes de o aplicativo iniciar a execução no depurador. O aplicativo efetua uma pausa quando a execução alcança o ponto de interrupção na linha 12. Nesse ponto, o depurador o notifica de que um ponto de interrupção foi alcançado e exibe o código-fonte nessa linha (12). Essa linha do código é a próxima instrução que será executada.
8. *Utilizando o comando* **`cont`** *para retomar a execução.* Digite `cont`. O **comando `cont`** faz com que o aplicativo continue a execução até o próximo ponto de interrupção ser alcançado (linha 19) quando o depurador o notifica (Figura N.6). Observe que a saída normal de `AccountTest` aparece entre as mensagens no depurador.
9. *Examinando o valor de uma variável.* Digite `print depositAmount` para exibir o valor atual armazenado na variável `depositAmount` (Figura N.7). O **comando `print`** permite examinar dentro do computador o valor de uma das suas variáveis. Esse comando o ajudará a encontrar e eliminar erros de lógica no seu código. Observe que o valor exibido é `25.0` — o valor atribuído a `depositAmount` na linha 15 da Figura N.1.
10. *Continuando a execução do aplicativo.* Digite `cont` para continuar a execução do aplicativo. Não há nenhum ponto de interrupção, portanto o aplicativo não está mais no modo de interrupção. O aplicativo continua a execução e conseqüentemente termina (Figura N.8). O depurador vai parar quando o aplicativo terminar.

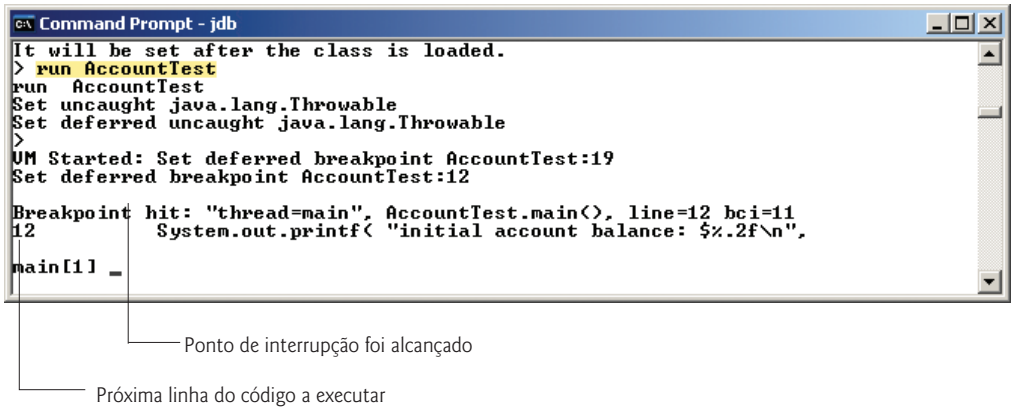


Figura N.5 Reiniciando o aplicativo `AccountTest`.

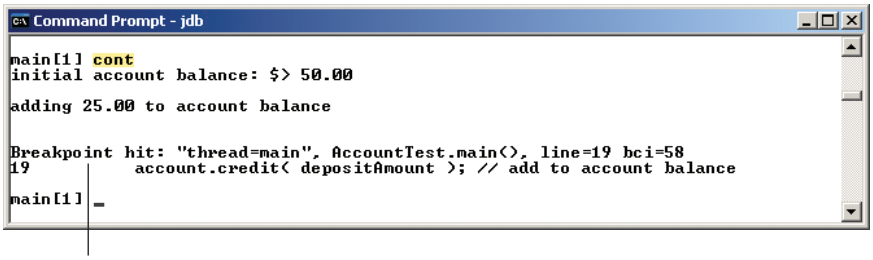


Figura N.6 A execução alcança o segundo ponto de interrupção.

Figura N.7 Examinando o valor da variável `depositAmount`.

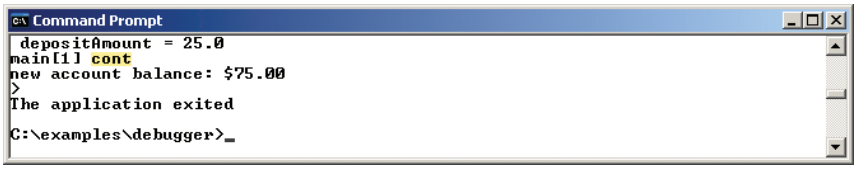


Figura N.8 Continuando a execução do aplicativo e encerrando o depurador.

Nesta seção, você aprendeu a ativar o depurador e a configurar pontos de interrupção para examinar variáveis com o comando `print` enquanto um aplicativo está em execução. Você também aprendeu a utilizar o comando `cont` para continuar a execução depois que um ponto de interrupção é alcançado.

N.3 Os comandos `print` e `set`

Na seção anterior, você aprendeu a utilizar o comando `print` do depurador para examinar o valor de uma variável durante a execução do programa. Nesta seção, aprenderá a utilizar o comando `print` para examinar o valor de expressões mais complexas. Você também aprenderá a utilizar o **comando `set`**, que permite ao programador atribuir novos valores a variáveis.

Para esta seção, supomos que você seguiu os *Passos 1* e *2*, da Seção N.2, para abrir a janela **Prompt do MS-DOS**, mudar para o diretório que contém os exemplos deste apêndice (por exemplo, `C:\examples\debugger`) e compilar o aplicativo `AccountTest` (e a classe `Account`) para depuração.

1. *Iniciando a depuração.* No **Prompt do MS-DOS**, digite `jdb` para iniciar o depurador Java.
2. *Inserindo um ponto de interrupção.* Configure um ponto de interrupção na linha 19 no código-fonte digitando `stop at AccountTest:19`.
3. *Executando o aplicativo e alcançando um ponto de interrupção.* Digite `run AccountTest` para começar o processo de depuração (Figura N.9). Isso fará com que o método `main` de `AccountTest` seja executado até que o ponto de interrupção na linha 19 seja alcançado. Isso suspende a execução do aplicativo e alterna o aplicativo para o modo de interrupção. Nesse ponto, as instruções nas linhas 9–13 criaram um objeto `Account` e imprimiram o saldo inicial do `Account` obtido chamando o método `getBalance`. A instrução na linha 15 (Figura N.1) declarou e inicializou a variável local `depositAmount` como `25.0`. A instrução na linha 19 é a próxima que será executada.
4. *Avaliando expressões aritméticas e booleanas.* Lembre-se, na Seção N.2, de que depois de o aplicativo entrar no modo de interrupção, você pode explorar os valores das variáveis do aplicativo utilizando o comando `print` do depurador. Você também pode usar o comando `print` para avaliar expressões aritméticas e `booleanas`. Na janela **Prompt do MS-DOS**, digite `print depositAmount - 2.0`. Observe que o comando `print` retorna o valor `23.0` (Figura N.10). Entretanto, esse comando na verdade não altera o valor de `depositAmount`. Na janela **Prompt do MS-DOS**, digite `print depositAmount == 23.0`. As expressões que contêm o símbolo `==` são tratadas como `booleanas`. O valor retornado é `false` (Figura N.10) porque `depositAmount` atualmente não contém o valor `23.0` — `depositAmount` ainda é `25.0`.
5. *Modificando valores.* O depurador permite alterar os valores das variáveis durante a execução do aplicativo. Isso pode ser valioso para experimentar diferentes valores e localizar erros de lógica nos aplicativos. Você pode utilizar o comando `set` do depurador para alterar o valor de uma variável. Digite `set depositAmount = 75.0`. O depurador altera o valor de `depositAmount` e exibe seu novo valor (Figura N.11).

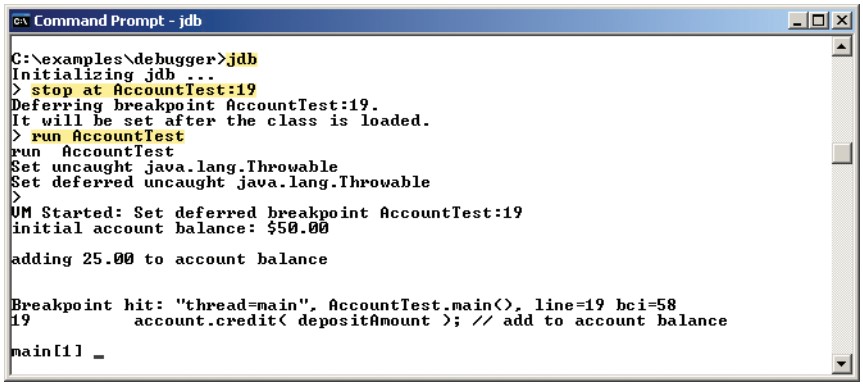


Figura N.9 A execução do aplicativo é suspensa quando o depurador alcança o ponto de interrupção na linha 19.

Figura N.10 Examinando os valores de uma expressão aritmética e `booleana`.

Figura N.11 Modificando valores.

6. *Visualizando o resultado do aplicativo.* Digite `cont` para continuar a execução do aplicativo. A linha 19 de `AccountTest` (Figura N.1) é executada, passando `depositAmount` para o método `credit` de `Account`. O método `main` então exibe o novo saldo. Observe que o resultado é `$125.00` (Figura N.12). Isso mostra que o passo anterior alterou o valor de `depositAmount` com base em seu valor inicial (`25.0`) para `75.0`.

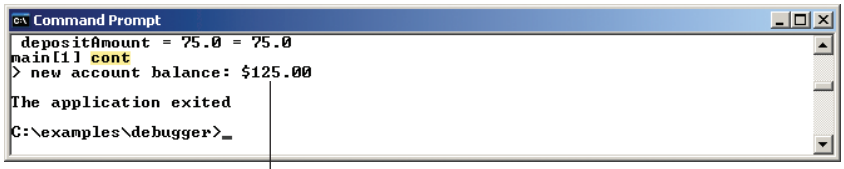
Nesta seção, você aprendeu a utilizar o comando `print` do depurador para avaliar expressões aritméticas e `booleanas`. Você também aprendeu a usar o comando `set` para modificar o valor de uma variável durante a execução do seu aplicativo.

N.4 Controlando a execução com os comandos `step`, `step up` e `next`

Às vezes, será necessário executar um aplicativo linha por linha para encontrar e corrigir erros. Investigar uma parte do seu aplicativo dessa maneira pode ajudá-lo a verificar se o código de um método executa corretamente. Nesta seção, você aprenderá a utilizar o depurador para essa tarefa. Os comandos que você aprenderá aqui permitem executar um método linha por linha, executar todas as instruções de um método de uma vez ou executar apenas as instruções remanescentes de um método (se você já tiver executado algumas instruções dentro do método).

Mais uma vez, supomos que você esteja trabalhando no diretório que contém os exemplos deste apêndice e que tenha compilado para depuração com a opção de compilador `-g`.

1. *Iniciando o depurador.* Inicie o depurador digitando `jdb`.
2. *Configurando um ponto de interrupção.* Digite `stop at AccountTest:19` para configurar um ponto de interrupção na linha 19.
3. *Executando o aplicativo.* Execute o aplicativo digitando `run AccountTest`. Depois de o aplicativo exibir as duas mensagens de saída, o depurador indica que o ponto de interrupção foi alcançado e exibe o código na linha 19 (Figura N.13). O depurador e o aplicativo então efetuam uma pausa e aguardam o próximo comando a ser inserido.
4. *Utilizando o comando `step`.* O **comando `step`** executa a próxima instrução no aplicativo. Se a próxima instrução a executar for uma chamada ao método, o controle será transferido para o método chamado. O comando `step` permite que você entre em um método e estude as instruções individuais dele. Por exemplo, você pode utilizar os comandos `print` e `set` para visualizar e modificar as variáveis dentro deles. Você agora utilizará o comando `step` para entrar no método `credit` da classe `Account` (Figura 3.13) digitando `step` (Figura N.14). O depurador indica que o passo foi completado e exibe a próxima instrução executável — nesse caso, a linha 21 da classe `Account` (Figura 3.13).



Novo valor do saldo da conta com base no valor alterado da variável `depositAmount`

Figura N.12 Saída exibida depois do processo de depuração.

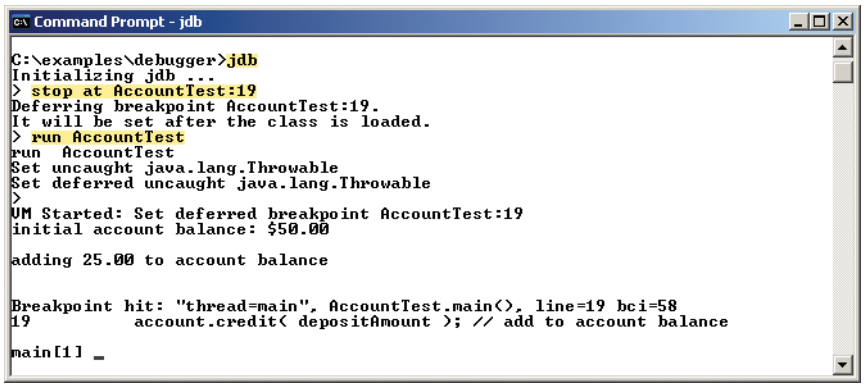


Figura N.13 Alcançando o ponto de interrupção no aplicativo `AccountTest`.

Figura N.14 Entrando na inspeção (*stepping into*) do método `credit`.

5. *Utilizando o comando `step up`.* Depois de utilizar o comando `step into` para inspecionar o método `credit`, digite **`step up`**. Esse comando executa as instruções remanescentes no método e retorna o controle ao local em que o método foi chamado. O método `credit` contém apenas uma instrução para adicionar o parâmetro `amount` do método à variável de instância `balance`. O comando `step up` executa essa instrução e então efetua uma pausa antes da linha 22 em `AccountTest`. Portanto, a próxima ação a ocorrer será imprimir o novo saldo da conta (Figura N.15). Em métodos longos, é recomendável examinar algumas linhas-chave do código e continuar a depurar o código do chamador. O comando `step up` é útil para situações em que você não quer continuar a investigar todo o método linha por linha.
6. *Utilizando o comando `cont` para continuar a execução.* Insira o comando `cont` (Figura N.16) para continuar a execução. A instrução nas linhas 22–23 é executada, exibindo o novo saldo e, então, o aplicativo e o depurador terminam.
7. *Reiniciando o depurador.* Reinicie o depurador digitando `jdb`.
8. *Configurando um ponto de interrupção.* Os pontos de interrupção só persistem até o final da sessão de depuração em que eles são configurados — depois de o depurador encerrar, todos os pontos de interrupção são removidos.

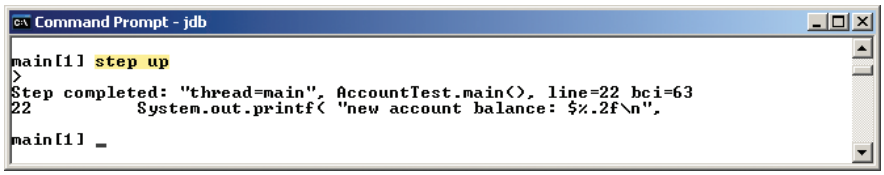


Figura N.15 Saindo da inspeção (*stepping out*) de um método.

(Na Seção N.6, você aprenderá a remover um ponto de interrupção manualmente antes do final da sessão de depuração.) Portanto, o ponto de interrupção configurado para a linha 19 no *Passo 2* não mais existirá depois da reinicialização do depurador no *Passo 7*. Para redefinir o ponto de interrupção na linha 19, digite mais uma vez `stop at AccountTest:19`.

9. *Executando o aplicativo.* Digite `run AccountTest` para executar o aplicativo. Como no *Passo 3*, `AccountTest` é executado até o ponto de interrupção na linha 19 ser alcançado, o depurador então pausa e espera o próximo comando (Figura N.17).

10. *Utilizando o comando `next`.* Digite **`next`**. Esse comando comporta-se como o comando `step`, exceto quando a próxima instrução a executar contém uma chamada de método. Nesse caso, o método chamado é executado na sua totalidade e o aplicativo avança para a próxima linha executável depois da chamada ao método (Figura N.18). Lembre-se, com base no que foi discutido no *Passo 4*, de que o comando `step` entraria no método chamado. Neste exemplo, o comando `next` faz com que método `Account` `credit` execute e, então, o depurador efetua uma pausa na linha 22 em `AccountTest`.

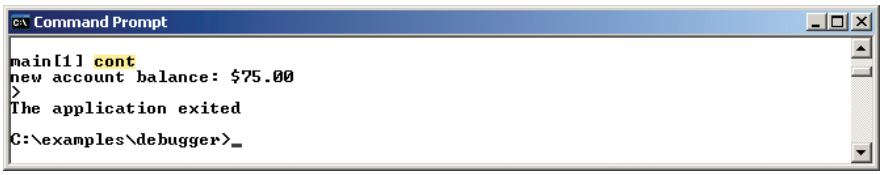


Figura N.16 Continuando a execução do aplicativo `AccountTest`.

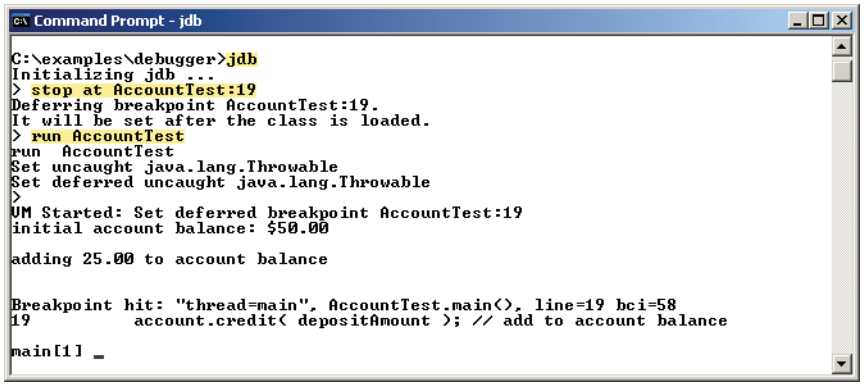


Figura N.17 Alcançando o ponto de interrupção no aplicativo `AccountTest`.

Figura N.18 Inspeção (*stepping over*) de uma chamada ao método.

11. *Utilizando o comando `exit`.* Utilize o **comando `exit`** para encerrar a sessão de depuração (Figura N.19). Esse comando faz com que o aplicativo `AccountTest` termine imediatamente em vez de executar as instruções restantes em `main`. Observe que ao depurar alguns tipos de aplicativos (por exemplo, aplicativos GUI), o aplicativo continua a executar mesmo depois de a sessão de depuração terminar.

Nesta seção, você aprendeu a utilizar os comandos `step` e `step up` do depurador para depurar métodos chamados durante a execução do seu aplicativo. Você viu como o comando `next` pode ser usado para inspecionar uma chamada ao método. Você também aprendeu que o comando `exit` encerra uma sessão de depuração.

N.5 O comando `watch`

Nesta seção, apresentamos o **comando `watch`**, que instrui o depurador a monitorar um campo. Quando esse campo está em vias de ser alterado, o depurador o notificará. Nesta seção, você aprenderá a utilizar o comando `watch` para ver como o campo `balance` do objeto `Account` é modificado durante a execução do aplicativo `AccountTest`.

Como ocorreu nas duas seções anteriores, supomos que você seguiu o *Passo 1* e o *Passo 2* na Seção N.2 para abrir a janela **Prompt do MS-DOS**, mudar para o diretório de exemplos correto e compilar as classes `AccountTest` e `Account` para depuração (isto é, com a opção de compilador `-g`).

1. *Iniciando o depurador.* Inicie o depurador digitando `jdb`.
2. *Monitorando um campo de uma classe.* Configure um ponto de monitoração (*watch*) no campo `balance` de `Account` digitando `watch Account.balance` (Figura N.20). Você pode configurar um ponto de monitoração em qualquer campo durante a execução do depurador. Sempre que o valor em um campo está em vias de mudar, o depurador entra no modo de interrupção e o notifica de que o valor mudará. Os pontos de monitoração só podem ser colocados em campos, não em variáveis locais.

3. *Executando o aplicativo*. Execute o aplicativo com o comando `run AccountTest`. O depurador agora o notificará de que o valor do campo `balance` irá mudar (Figura N.21). Quando o aplicativo é inicializado, uma instância de `Account` é criada com um saldo inicial de US$ 50.00 e uma referência ao objeto `Account` é atribuído à variável local `account` (linha 9, Figura N.1). Lembre-se, na Figura 3.13, de que quando o construtor desse objeto é executado, se o parâmetro `initialBalance` for maior que `0.0`, a variável de instância `balance` é atribuída ao valor do parâmetro `initialBalance`. O depurador o notifica de que o valor de `balance` será configurado como `50.0`.

Figura N.19 Encerrando o depurador.

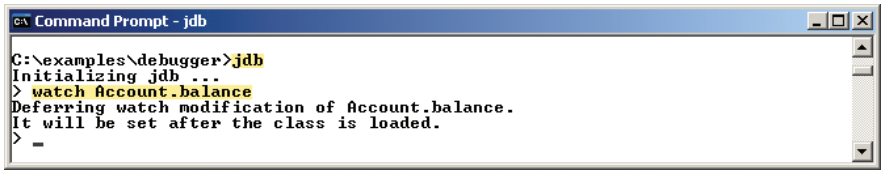


Figura N.20 Configurando um watch no campo `balance` de `Account`.

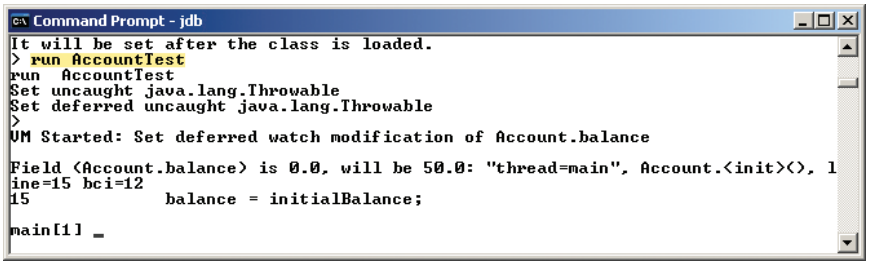


Figura N.21 O aplicativo `AccountTest` pára quando `account` é criado e seu campo `balance` modificado.

4. *Adicionando dinheiro à conta*. Digite `cont` para continuar a executar o aplicativo. O aplicativo executa normalmente antes de alcançar o código na linha 19 da Figura N.1 que chama o método `credit` de `Account` para aumentar o `balance` do objeto `Account` por um `amount` especificado. O depurador o notifica de que a variável de instância `balance` mudará (Figura N.22). Observe que embora a linha 19 da classe `AccountTest` chame o método `credit`, é a linha 21 no método `credit` de `Account` que na verdade altera o valor de `balance`.
5. *Continuando a execução*. Digite `cont` — o aplicativo terminará a execução por não tentar nenhuma alteração adicional em `balance` (Figura N.23).
6. *Reiniciando o depurador e redefinindo o ponto de monitoração sobre a variável*. Digite `jdb` para reiniciar o depurador. Mais uma vez, configure um ponto de monitoração na variável de instância `balance` de `Account` digitando `watchAccount.balance` e, então, digite `run AccountTest` para executar o aplicativo (Figura N.24).

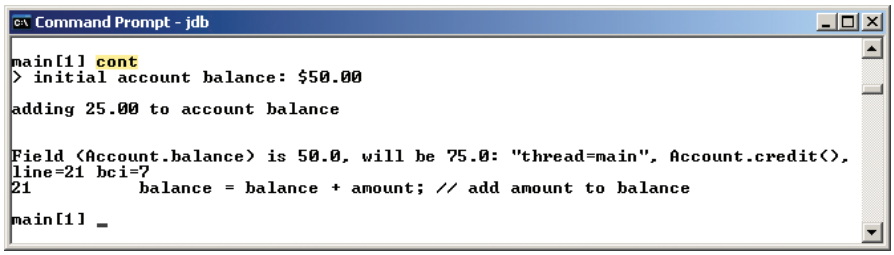


Figura N.22 Alterando o valor de `balance` chamando o método `credit` de `Account`.

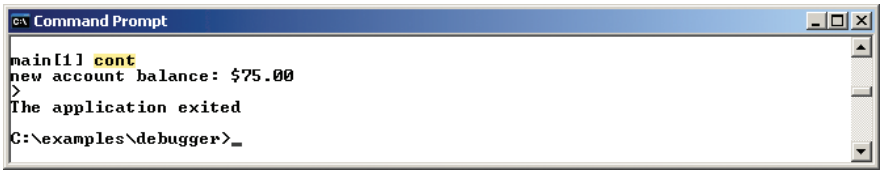


Figura N.23 Continuando a execução de `AccountTest`.

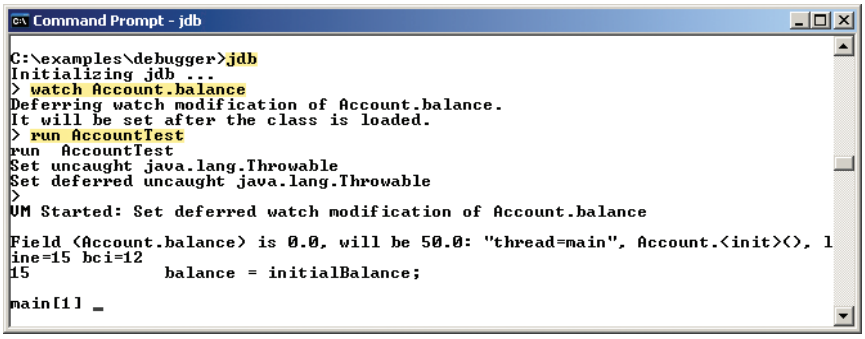


Figura N.24 Reiniciando o depurador e redefinindo o ponto de monitoração na variável `balance`.

7. *Removendo o ponto de monitoração no campo.* Suponha que em um campo você queira monitorar somente uma parte da execução de um programa. Você pode remover o ponto de monitoração do depurador na variável `balance` digitando **`unwatch`** `Account.balance` (Figura N.25). Digite `cont` — o aplicativo terminará a execução sem reentrar no modo de interrupção.
8. *Fechando a janela* ***Prompt do MS-DOS****.* Feche a janela **Prompt do MS-DOS** clicando no botão de fechar.

Nesta seção, você aprendeu a utilizar o comando `watch` para ativar o depurador a fim de notificá-lo sobre alterações no valor de um campo por todo o ciclo de vida de um aplicativo. Você também aprendeu a utilizar o comando `unwatch` para remover um monitor (watch) de um campo antes do fim do aplicativo.

N.6 O comando `clear`

Na seção anterior, você aprendeu a utilizar o comando `unwatch` para remover um monitor (watch) de um campo. O depurador também fornece o comando `clear` para remover um ponto de interrupção de um aplicativo. Freqüentemente, você precisará depurar aplicativos que contêm ações repetitivas, como um loop. Talvez você queira examinar os valores das variáveis durante várias, mas possivelmente não todas, iterações do loop. Se você configurar um ponto de interrupção no corpo de um loop, o depurador efetuará uma pausa antes de cada execução da linha que contém um ponto de interrupção. Depois de determinar que o loop está funcionando adequadamente, talvez você queira remover o ponto de interrupção e permitir que as iterações restantes prossigam normalmente. Nesta seção, utilizaremos o aplicativo de juros compostos da Figura 5.6 para demonstrar como o depurador comporta-se quando você configura um ponto de interrupção no corpo de uma instrução `for` e como remover um ponto de interrupção no meio de uma sessão de depuração.

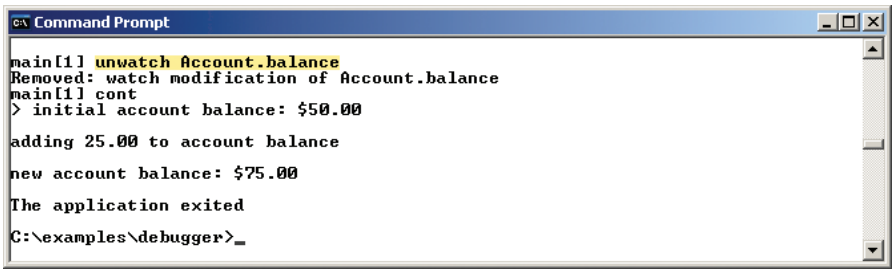


Figura N.25 Removendo o ponto de monitoração na variável `balance`.

1. *Abrindo a janela* **Prompt do MS-DOS***, mudando diretórios e compilando o aplicativo para depuração.* Abra a janela **Prompt do MS-DOS** e então mude para o diretório que contém os exemplos deste apêndice. Para sua conveniência, fornecemos uma cópia do arquivo `Interest.java` neste diretório. Compile o aplicativo para depuração digitando `javac -g Interest.java`.
2. *Iniciando o depurador e configurando pontos de interrupção.* Inicie o depurador digitando `jdb`. Configure os pontos de interrupção nas linhas 13 e 22 da classe `Interest` digitando `stop at Interest:13` e então `stop at Interest:22` (Figura N.26).

3. *Executando o aplicativo.* Execute o aplicativo digitando `run Interest`. O aplicativo executa até alcançar o ponto de interrupção na linha 13 (Figura N.27).
4. *Continuando a execução.* Digite `cont` para continuar — o aplicativo executa a linha 13, imprimindo os títulos da coluna `"Year"` e `"Amount on deposit"`. Observe que a linha 13 aparece antes da instrução `for` nas linhas 16–23 em `Interest` (Figura 5.6) e assim é executada somente uma vez. A execução continua depois da linha 13 até o ponto de interrupção na linha 22 ser alcançado durante a primeira iteração da instrução `for` (Figura N.28).
5. *Examinando valores de variáveis.* Digite `print year` para examinar o valor atual da variável `year` (a variável de controle do `for`). Imprima também o valor da variável `amount` (Figura N.29).
6. *Continuando a execução.* Digite `cont` para continuar a execução. A linha 22 executa e imprime os valores atuais de `year` e `amount`. Depois de o `for` entrar na sua segunda iteração, o depurador o notifica de que o ponto de interrupção na linha 22 foi alcançado uma segunda vez. Observe que o depurador efetua uma pausa toda vez que a linha em que um ponto de interrupção foi configurado está em vias de executar — quando o ponto de interrupção aparece em um loop, o depurador efetua uma pausa durante cada iteração. Imprima os valores das variáveis `year` e `amount` mais uma vez para ver como os valores mudaram desde a primeira iteração do `for` (Figura N.30).

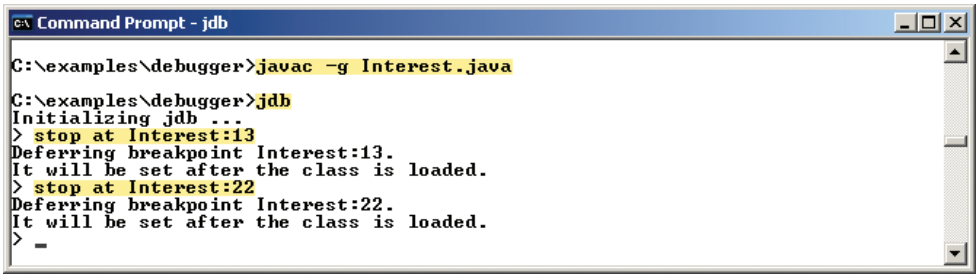


Figura N.26 Configurando pontos de interrupção no aplicativo `Interest`.

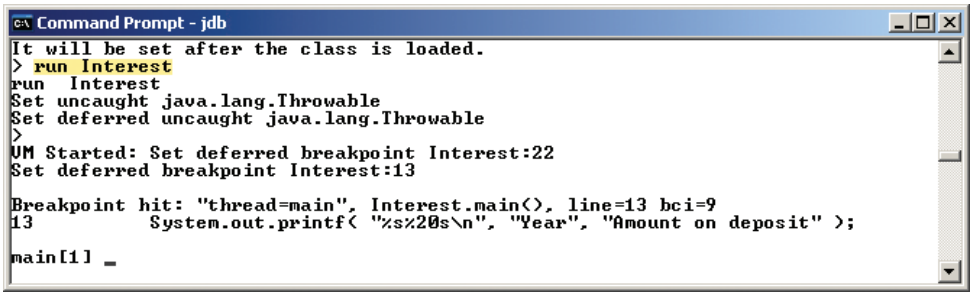


Figura N.27 Alcançando o ponto de interrupção na linha 13 no aplicativo `Interest`.

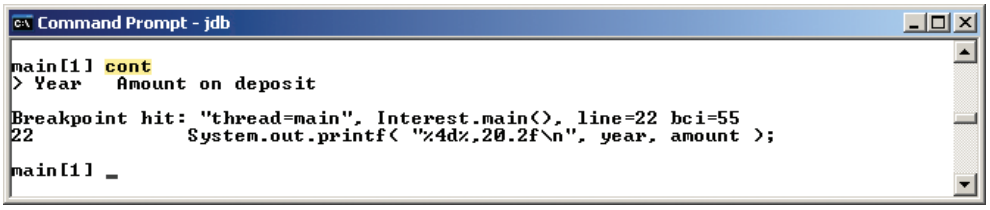


Figura N.28 Alcançando o ponto de interrupção na linha 22 no aplicativo `Interest`.

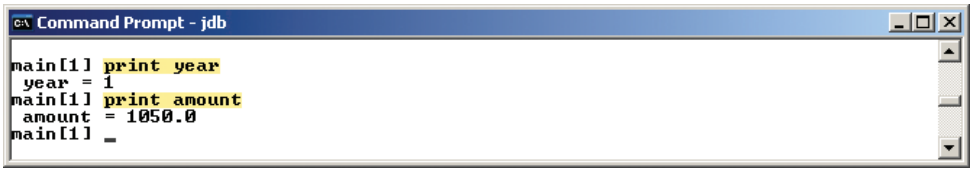


Figura N.29 Imprimindo `year` e `amount` durante a primeira iteração do `for` de `Interest`.

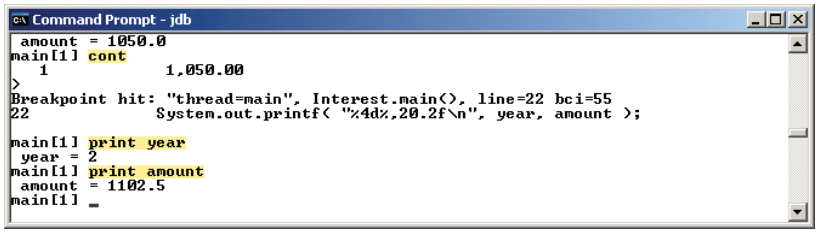


Figura N.30 Imprimindo `year` e `amount` durante a segunda iteração do `for` de `Interest`.

7. *Removendo um ponto de interrupção.* Você pode exibir uma lista de todos os pontos de interrupção no aplicativo digitando **`clear`** (Figura N.31). Suponha que você esteja satisfeito com o funcionamento da instrução `for` do aplicativo `Interest`, assim você quer remover o ponto de interrupção na linha 22 e permitir que as demais iterações do loop prossigam normalmente. Você pode remover o ponto de interrupção na linha 22 digitando `clear Interest:22`. Agora digite `clear` para listar os pontos de interrupção restantes no aplicativo. O depurador deve indicar que só o ponto de interrupção na linha 13 permanece (Figura N.31). Observe que esse ponto de interrupção já foi alcançado e assim não mais afetará a execução.
8. *Continuando a execução depois de remover um ponto de interrupção.* Digite `cont` para continuar a execução. Lembre-se de que a execução sofreu uma pausa pela última vez antes da instrução `printf` na linha 22. Se o ponto de interrupção na linha 22 foi removido com sucesso, continuar o aplicativo gerará a saída correta para as iterações atuais e remanescentes da instrução `for` sem que o aplicativo pare (Figura N.32).

Figura N.31 Removendo o ponto de interrupção na linha 22.

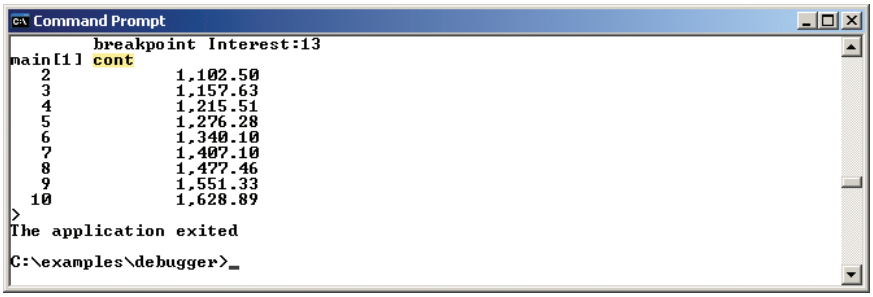


Figura N.32 O aplicativo executa sem um ponto de interrupção configurado na linha 22.

Nesta seção, você aprendeu a utilizar o comando `clear` para listar todos os pontos de interrupção configurados para um aplicativo e a remover um ponto de interrupção.

N.7 Conclusão

Neste apêndice, você aprendeu a inserir e remover pontos de interrupção no depurador. Os pontos de interrupção permitem pausar a execução do aplicativo para você poder examinar os valores das variáveis com o comando `print` do depurador. Essa capacidade o ajudará a localizar e corrigir erros de lógica nos seus aplicativos. Você viu como utilizar o comando `print` para examinar o valor de uma expressão e como usar o comando `set` para alterar o valor de uma variável. Você também aprendeu os comandos de depurador (incluindo `step`, `step up` e `next`) que podem ser utilizados para determinar se um método está executando corretamente. Você aprendeu a utilizar o comando `watch` para monitorar um campo por toda a vida de um aplicativo. Por fim, você aprendeu a usar o comando `clear` para listar todos os pontos de interrupção configurados para um aplicativo ou a remover pontos de interrupção individuais para continuar a execução sem pontos de interrupção.

Resumo

- O depurador permite monitorar a execução de um aplicativo para você poder localizar e remover erros de lógica.
- A opção de compilador `-g` compila uma classe para depuração.
- O comando `jdb` inicia o depurador.
- O comando `run`, seguido pelo nome da classe de um aplicativo, executa o aplicativo por meio do depurador.
- O comando `stop`, seguido pelo nome da classe, de dois-pontos e de um número da linha, configura um ponto de interrupção no número da linha especificado.
- O comando `cont` retoma a execução depois de entrar no modo de interrupção.
- O comando `print`, seguido pelo nome de uma variável, examina o conteúdo da variável especificada.
- O comando `print` pode ser usado para examinar o valor de uma expressão durante a execução de um aplicativo.
- O comando `set` modifica o valor de uma variável durante a execução de um aplicativo.
- O comando `step` executa a próxima instrução no aplicativo. Se a próxima instrução a executar for uma chamada ao método, o controle será transferido para o método chamado.
- O comando `step up` executa as instruções em um método e retorna o controle ao local em que o método foi chamado.
- O comando `next` executa a próxima instrução no aplicativo. Se a próxima instrução a executar for uma chamada ao método, o método chamado executará na sua totalidade (sem transferir o controle e inserir o método) e o aplicativo passará para a próxima linha executável depois da chamada ao método.
- O comando `watch` instrui o depurador a notificá-lo se o campo especificado for modificado.
- O comando `unwatch` remove um watch de um campo.
- O comando `clear`, executado por si só, lista os pontos de interrupção configurados para um aplicativo.
- O comando `clear`, seguido por um nome de classe, de dois-pontos e de um número de linha, remove o ponto de interrupção especificado.

Terminologia

`clear`, comando
`cont`, comando
depurador
`exit`, comando
`-g`, opção de compilador
`jdb`, comando
modo de interrupção
`next`, comando
ponto de interrupção
`print`, comando
`run`, comando
`set`, comando
`step up`, comando
`step`, comando
`stop`, comando
`unwatch`, comando
`watch`, comando

Exercícios de revisão

N.1 Preencha as lacunas em cada uma das seguintes afirmações:

a) Um ponto de interrupção não pode ser configurado em um(a) _____.
b) Você pode examinar o valor de uma expressão utilizando o comando _____ do depurador.
c) Você pode modificar o valor de uma variável utilizando o comando _____ do depurador.
d) Durante a depuração, o comando _____ executa as instruções restantes no método atual e retorna o controle do programa ao local em que o método foi chamado.
e) O comando _____ do depurador comporta-se como o comando `step` quando a próxima instrução a executar não contém uma chamada ao método.
f) O comando `watch` do depurador permite visualizar todas as alterações em um(a) _____.

N.2 Determine se cada um dos seguintes itens é *verdadeiro* ou *falso*. Se *falso*, explique por quê.

a) Quando a execução do aplicativo é suspensa em um ponto de interrupção, a próxima instrução a ser executada é a instrução depois do ponto de interrupção.
b) Os comandos `watch` podem ser removidos com o comando `clear` do depurador.
c) A opção de compilador `-g` deve ser usada ao compilar classes para depuração.
d) Quando um ponto de interrupção aparece em um loop, o depurador só faz uma pausa na primeira vez em que o ponto de interrupção é encontrado.

Respostas dos exercícios de revisão

N.1 a) comentário. b) `print`. c) `set`. d) `step up`. e) `next`. f) campo.

N.2 a) Falso. Quando a execução do aplicativo é suspensa em um ponto de interrupção, a próxima instrução a ser executada é a instrução no ponto de interrupção. b) Falso. Os comandos `watch` podem ser removidos com o comando `unwatch` do depurador. c) Verdadeiro. d) Falso. Quando um ponto de interrupção aparece em um loop, o depurador faz uma pausa durante cada iteração.