```java
// Figura 3.1: Account.java
// Classe Account que contém uma variável de instância name
// e métodos para configurar e obter seu valor.

public class Account
{
   private String name; // variável de instância

   // método para definir o nome no objeto
   public void setName(String name)
   {
      this.name = name; // armazena o nome
   }

   // método para recuperar o nome do objeto
   public String getName()
   {
      return name; // retorna valor do nome para o chamador
   }
} // fim da classe Account
```

```java
// Figura 3.2: AccountTest.Java
// Cria e manipula um objeto Account.
import java.util.Scanner;

public class AccountTest
{
   public static void main(String[] args)
   {
      // cria um objeto Scanner para obter entrada a partir da janela de comando
      Scanner input = new Scanner(System.in);

      // cria um objeto Account e o atribui a myAccount
      Account myAccount = new Account();

      // exibe o valor inicial do nome (null)
      System.out.printf("Initial name is: %s%n%n", myAccount.getName());

      // solicita e lê o nome
      System.out.println("Please enter the name:");
      String theName = input.nextLine(); // lê uma linha de texto
      myAccount.setName(theName); // insere theName em myAccount
      System.out.println(); // gera saída de uma linha em branco

      // exibe o nome armazenado no objeto myAccount
      System.out.printf("Name in object myAccount is:%n%s%n",
         myAccount.getName());
   }
} // fim da classe AccountTest
```

```
Initial name is: null
Please enter the name:
Jane Green
Name in object myAccount is:
Jane Green
```

Compilação e execução de um aplicativo com múltiplas classes

```
javac Account.java AccountTest.java
```

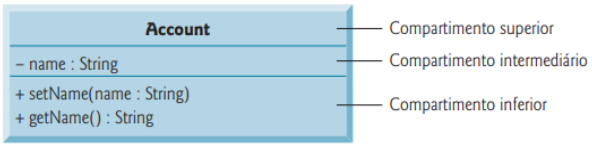
para compilar *ambas* as classes de uma vez. Se o diretório que contém o aplicativo incluir *apenas* os arquivos desse aplicativo, você pode compilar ambas as classes com o comando

```
javac *.java
```

O asterisco (*) em `*.java` indica que *todos* os arquivos no diretório *atual* que têm a extensão de nome de arquivo "`.java`" devem ser compilados. Se ambas as classes forem compiladas corretamente — isto é, nenhum erro de compilação for exibido — você pode então executar o aplicativo com o comando

```
java AccountTest
```

Diagrama de classe UML de `Account` com uma variável de instância e os métodos *set* e *get*



Compartimento superior

Na UML, cada classe é modelada em um diagrama de classe como um retângulo com três compartimentos. Nesse diagrama, o compartimento *superior* contém o *nome da classe* `Account` centralizado horizontalmente em negrito.

Compartimento intermediário

O compartimento *intermediário* contém o *atributo* `name` *da classe*, que corresponde à variável de instância de mesmo nome em Java. A variável de instância `name` é `private` em Java, assim o diagrama UML de classe lista um *modificador de acesso com um sinal de subtração (–)* antes do nome do atributo. Depois do nome do atributo há um *dois pontos* e o *tipo de atributo,* nesse caso `String`.

Compartimento inferior

O compartimento *inferior* contém as **operações** da classe, `setName` e `getName`, que correspondem aos métodos com os mesmos nomes em Java. O UML modela as operações listando o nome de cada uma precedido por um *modificador de acesso,* nesse caso `+ getName`. Esse sinal de adição (+) indica que `getName` é uma operação *pública* na UML (porque é um método `public` em Java). A operação `getName` *não* tem nenhum parâmetro, então os parênteses após o nome dela no diagrama de classe estão *vazios,* assim como na declaração de método na linha 16 da Figura 3.1. A operação `setName`, também de caráter público, tem um parâmetro `String` chamado `name`.

Tipos de retorno

A UML indica o *tipo de retorno* de uma operação inserindo dois pontos e o tipo de retorno *após* os parênteses que vêm depois do nome da operação. O método `getName` de `Account` (Figura 3.1) tem um tipo de retorno `String`. O método `setName` *não* retorna um valor (porque retorna `void` em Java), então o diagrama de classe UML *não* especifica um tipo de retorno após os parênteses dessa operação.

Parâmetros

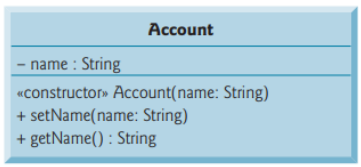
A UML modela um parâmetro de um modo pouco diferente do Java listando o nome desse parâmetro, seguido por dois-pontos e pelo tipo dele nos parênteses que seguem o nome da operação. O UML tem seus próprios tipos de dado semelhantes àqueles do Java, mas, para simplificar, usaremos os tipos de dado Java. O método `setName` de `Account` (Figura 3.1) tem um parâmetro `String` chamado `name`, assim a Figura 3.3 lista `name : String` entre parênteses após o nome do método.

```
1  // Figura 3.5: Account.java
2  // a classe Account com um construtor que inicializa o nome.
3
4  public class Account
5  {
6     private String name; // variável de instância
7
8     // o construtor inicializa name com nome do parâmetro
9     public Account(String name) // o nome do construtor é nome da classe
10    {
11       this.name = name;
12    }
13
14    // método para configurar o nome
15    public void setName(String name)
16    {
17       this.name = name;
18    }
19
20    // método para recuperar o nome do curso
21    public String getName()
22    {
23       return name;
24    }
25 } // fim da classe Account
```

```java
// Figura 3.6: AccountTest.Java
// Usando o construtor de Account para inicializar a instância name
// variável no momento em que cada objeto Account é criado.

public class AccountTest
{
   public static void main(String[] args)
   {
      // cria dois objetos Account
      Account account1 = new Account("Jane Green");
      Account account2 = new Account("John Blue");

      // exibe o valor inicial de nome para cada Account
      System.out.printf("account1 name is: %s%n", account1.getName());
      System.out.printf("account2 name is: %s%n", account2.getName());
   }
} // fim da classe AccountTest
```

```
account1 name is: Jane Green
account2 name is: John Blue
```

Adicionando o construtor ao diagrama de classe UML da classe `Account`



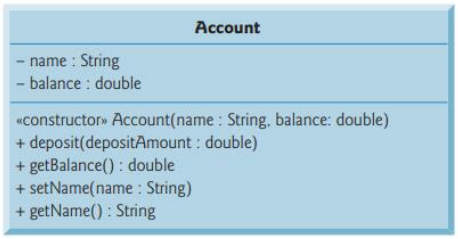

```java
// Figura 3.8: Account.java
// Classe Account com uma variável de instância balance do tipo double e um construtor
// e método deposit que executa a validação.

public class Account
{
   private String name; // variável de instância
   private double balance; // variável de instância

   // Construtor de Account que recebe dois parâmetros
   public Account(String name, double balance)
   {
      this.name = name; // atribui name à variável de instância name

      // valida que o balance é maior que 0.0; se não for,
      // a variável de instância balance mantém seu valor inicial padrão de 0.0
      if (balance > 0.0) // se o saldo for válido
         this.balance = balance; // o atribui à variável de instância balance
   }

   // método que deposita (adiciona) apenas uma quantia válida no saldo
   public void deposit(double depositAmount)
   {
      if (depositAmount > 0.0) // se depositAmount for válido
         balance = balance + depositAmount; // o adiciona ao saldo
   }
```

```
27
28      // método retorna o saldo da conta
29      public double getBalance()
30      {
31          return balance;
32      }
33
34      // método que define o nome
35      public void setName(String name)
36      {
37         this.name = name;
38      }
39
40      // método que retorna o nome
41      public String getName()
42      {
43         return name; // retorna o valor de name ao chamador
44      } // fim do método getName
45   } // fim da classe Account
```

```
1    // Figura 3.9: AccountTest.Java
2    // Entrada e saída de números de ponto flutuante com objetos Account.
3    import java.util.Scanner;
4
5    public class AccountTest
6    {
7       public static void main(String[] args)
8       {
9          Account account1 = new Account("Jane Green", 50.00);
10         Account account2 = new Account("John Blue", -7.53);
11
12         // exibe saldo inicial de cada objeto
13         System.out.printf("%s balance: $%.2f %n",
14            account1.getName(),account1.getBalance());
15         System.out.printf("%s balance: $%.2f %n%n",
16            account2.getName(),account2.getBalance());
17
18         // cria um Scanner para obter entrada a partir da janela de comando
19         Scanner input = new Scanner(System.in);
20
21         System.out.print("Enter deposit amount for account1: "); // prompt
22         double depositAmount = input.nextDouble(); // obtém a entrada do usuário
23         System.out.printf("%nadding %.2f to account1 balance%n%n",
24            depositAmount);
25         account1.deposit(depositAmount); // adiciona o saldo de account1
26
27         // exibe os saldos
28         System.out.printf("%s balance: $%.2f %n",
29            account1.getName(),account1.getBalance());
30         System.out.printf("%s balance: $%.2f %n%n",
31            account2.getName(),account2.getBalance());
32
33         System.out.print("Enter deposit amount for account2: "); // prompt
34         depositAmount = input.nextDouble(); // obtém a entrada do usuário
35         System.out.printf("%nadding %.2f to account2 balance%n%n",
36            depositAmount);
37         account2.deposit(depositAmount); // adiciona ao saldo de account2
38
39         // exibe os saldos
40         System.out.printf("%s balance: $%.2f %n",
41            account1.getName(),account1.getBalance());
42         System.out.printf("%s balance: $%.2f %n%n",
43            account2.getName(),account2.getBalance());
44      } // fim de main
45   } // fim da classe AccountTest
```

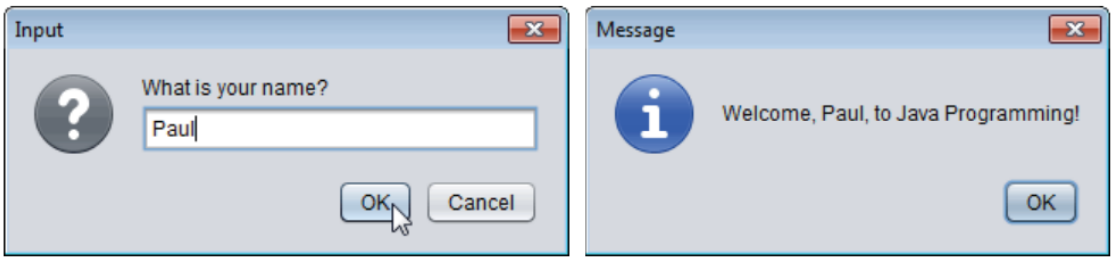
```
Jane Green balance: $50.00
John Blue balance: $0.00
Enter deposit amount for account1: 25.53
adding 25.53 to account1 balance
Jane Green balance: $75.53
John Blue balance: $0.00

Enter deposit amount for account2: 123.45
adding 123.45 to account2 balance
Jane Green balance: $75.53
John Blue balance: $123.45
```

Diagrama de classe UML para a classe Account




```java
// Figura 3.12: Dialog1.java
// Usando JOptionPane para exibir múltiplas linhas em uma caixa de diálogo.
import javax.swing.JOptionPane;

public class Dialog1
{
   public static void main(String[] args)
   {
      // exibe um diálogo com uma mensagem
      JOptionPane.showMessageDialog(null, "Welcome to Java");
   }
} // fim da classe Dialog1
```


```java
// Figura 3.13: NameDialog.Java
// Obtendo a entrada de usuário a partir de um diálogo.
import javax.swing.JOptionPane;

public class NameDialog
{
   public static void main(String[] args)
   {
      // pede para o usuário inserir seu nome
      String name = JOptionPane.showInputDialog("What is your name?");

      // cria a mensagem
      String message =
         String.format("Welcome, %s, to Java Programming!", name);

      // exibe a mensagem para cumprimentar o usuário pelo nome
      JOptionPane.showMessageDialog(null, message);
   } // fim de main
} // termina NameDialog
```


Input
What is your name?
Paul
OK
Cancel
Message
Welcome, Paul, to Java Programming!
OK