

3

Introduction to Classes and Objects



You will see something new. Two things. And I call them Thing One and Thing Two.

— Dr. Theodor Seuss Geisel

Nothing can have value without being an object of utility.

— Karl Marx

Your public servants serve you right.

— Adlai E. Stevenson

*Knowing how to answer one who speaks,
To reply to one who sends a message.*

— Amenemope



OBJECTIVES

In this chapter you will learn:

- What classes, objects, methods and instance variables are.
- How to declare a class and use it to create an object.
- How to declare methods in a class to implement the class's behaviors.
- How to declare instance variables in a class to implement the class's attributes.
- How to call an object's method to make that method perform its task.
- The differences between instance variables of a class and local variables of a method.
- How to use a constructor to ensure that an object's data is initialized when the object is created.
- The differences between primitive and reference types.



Outline

- 3.1 Introduction**
- 3.2 Classes, Objects, Methods and Instance Variables**
- 3.3 Declaring a Class with a Method and Instantiating an Object of a Class**
- 3.4 Declaring a Method with a Parameter**
- 3.5 Instance Variables, *set* Methods and *get* Methods**
- 3.6 Primitive Types vs. Reference Types**
- 3.7 Initializing Objects with Constructors**
- 3.8 Floating-Point Numbers and Type `double`**



3.1 Introduction

- **Classes**
- **Floating-Point numbers**



3.2 Classes, Objects, Methods and Instance Variables

- **Class provides one or more methods**
- **Method represents task in a program**
 - **Describes the mechanisms that actually perform its tasks**
 - **Hides from its user the complex tasks that it performs**
 - **Method call tells method to perform its task**



3.2 Classes, Objects, Methods and Instance Variables (Cont.)

- **Classes contain one or more attributes**
 - Specified by instance variables
 - Carried with the object as it is used



3.3 Declaring a Class with a Method and Instantiating an Object of a Class

- Each class declaration that begins with keyword `public` must be stored in a file that has the same name as the class and ends with the `.java` file-name extension.



Class GradeBook

- keyword `public` is an access modifier
- Class declarations include:
 - Access modifier
 - Keyword `class`
 - Pair of left and right braces



Class GradeBook

- **Method declarations**
 - Keyword `public` indicates method is available to public
 - Keyword `void` indicates no return type
 - Access modifier, return type, name of method and parentheses comprise method header



Common Programming Error 3.1

Declaring more than one `public` class in the same file is a compilation error.



Outline

GradeBook.java

```
1 // Fig. 3.1: GradeBook.java
2 // Class declaration with one method.
3
4 public class GradeBook
5 {
6     // display a welcome message to the GradeBook
7     public void displayMessage()
8     {
9         System.out.println( "Welcome to the Grade Book!" );
10    } // end method displayMessage
11
12 } // end class GradeBook
```

Print line of text to output



Class GradeBookTest

- **Java is extensible**
 - Programmers can create new classes
- **Class instance creation expression**
 - Keyword `new`
 - Then name of class to create and parentheses
- **Calling a method**
 - Object name, then dot separator (`.`)
 - Then method name and parentheses



Outline

GradeBookTest.java

```
1 // Fig. 3.2: GradeBookTest.java
2 // Create a GradeBook object and call its displayMessage method.
3
4 public class GradeBookTest
5 {
6     // main method begins program execution
7     public static void main( String args[] )
8     {
9         // create a GradeBook object and assign it
10        GradeBook myGradeBook = new GradeBook();
11
12        // call myGradeBook's displayMessage method
13        myGradeBook.displayMessage();
14    } // end main
15
16 } // end class GradeBookTest
```

Use class instance creation
expression to create object of class
GradeBook

Call method displayMessage
using GradeBook object

Welcome to the Grade Book!



Compiling an Application with Multiple Classes

- **Compiling multiple classes**
 - List each `.java` file in the compilation command and separate them with spaces
 - Compile with `*.java` to compile all `.java` files in that directory



UML Class Diagram for Class GradeBook

- **UML class diagrams**
 - **Top compartment contains name of the class**
 - **Middle compartment contains class's attributes or instance variables**
 - **Bottom compartment contains class's operations or methods**
 - **Plus sign indicates public methods**



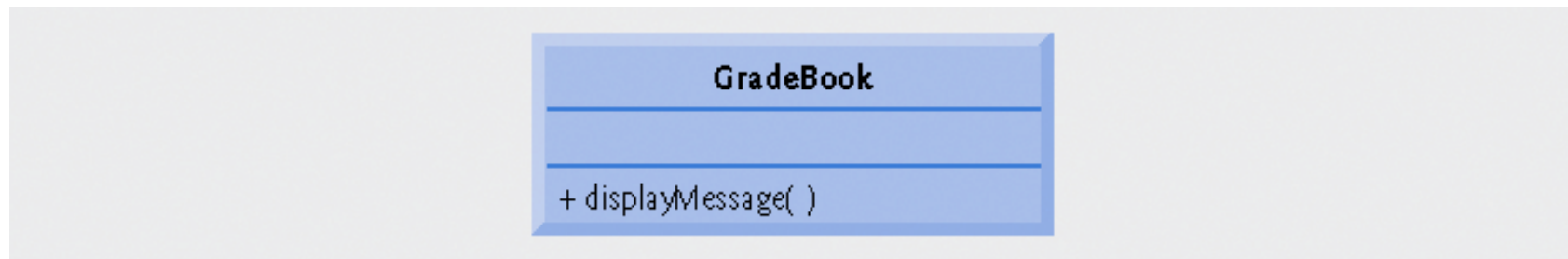


Fig. 3.3 | UML class diagram indicating that class GradeBook has a public `displayMessage` operation.



3.4 Declaring a Method with a Parameter

- **Method parameters**
 - Additional information passed to a method
 - Supplied in the method call with arguments



3.4 Declaring a Method with a Parameter

- **Scanner methods**
 - `nextLine` reads next line of input
 - `next` reads next word of input




Outline

GradeBook.java

```
1 // Fig. 3.4: GradeBook.java
2 // Class declaration with a method that has a parameter.
3
4 public class GradeBook
5 {
6     // display a welcome message to the GradeBook user
7     public void displayMessage( String courseName )
8     {
9         System.out.printf( "welcome to the grade book for\n%s!\n",
10             courseName );
11     } // end method displayMessage
12
13 } // end class GradeBook
```

Call printf method with
courseName argument



Outline

GradeBookTest.java

```
1 // Fig. 3.5: GradeBookTest.java
2 // Create GradeBook object and pass a String to
3 // its displayMessage method.
4 import java.util.Scanner; // program uses Scanner
5
6 public class GradeBookTest
7 {
8     // main method begins program execution
9     public static void main( String args[] )
10    {
11        // create Scanner to obtain input from command window
12        Scanner input = new Scanner( System.in );
13
14        // create a GradeBook object and assign it
15        GradeBook myGradeBook = new GradeBook();
16
17        // prompt for and input course name
18        System.out.println( "Please enter the course name:" );
19        String nameOfCourse = input.nextLine(); //
20        System.out.println(); // outputs a blank line
21
22        // call myGradeBook's displayMessage method
23        // and pass nameOfCourse as an argument
24        myGradeBook.displayMessage( nameOfCourse );
25    } // end main
26
27 } // end class GradeBookTest
```

Call `nextLine` method to read a line of input

Call `displayMessage` with an argument

```
Please enter the course name:
CS101 Introduction to Java Programming
```

```
welcome to the grade book for
CS101 Introduction to Java Programming!
```



Software Engineering Observation 3.1

Normally, objects are created with new. One exception is a string literal that is contained in quotes, such as "hello". String literals are references to String objects that are implicitly created by Java.



More on Arguments and Parameters

- **Parameters specified in method's parameter list**
 - Part of method header
 - Uses a comma-separated list



Common Programming Error 3.2

A compilation error occurs if the number of arguments in a method call does not match the number of parameters in the method declaration.



Common Programming Error 3.3

A compilation error occurs if the types of the arguments in a method call are not consistent with the types of the corresponding parameters in the method declaration.



Updated UML Class Diagram for Class GradeBook

- **UML class diagram**
 - Parameters specified by parameter name followed by a colon and parameter type



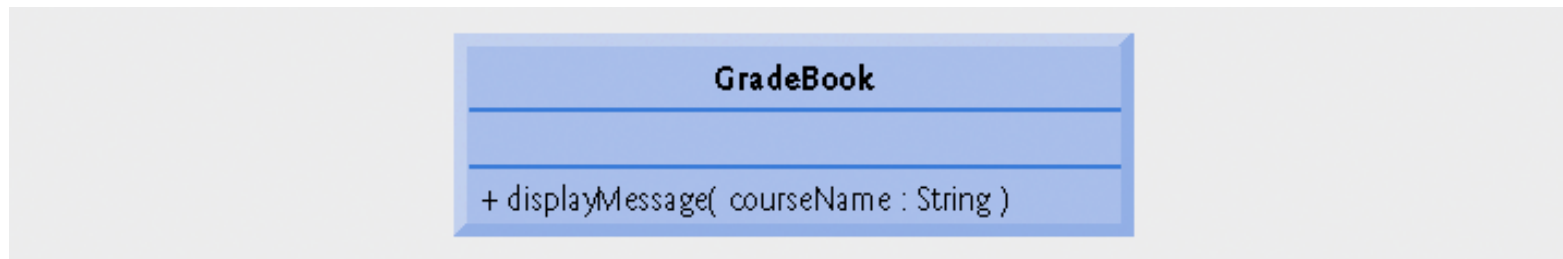


Fig. 3.6 | UML class diagram indicating that class `GradeBook` has a `displayMessage` operation with a `courseName` parameter of UML type `String`.



Notes on Import Declarations

- **java.lang is implicitly imported into every program**
- **Default package**
 - Contains classes compiled in the same directory
 - Implicitly imported into source code of other files in directory
- **Imports unnecessary if fully-qualified names are used**



Software Engineering Observation 3.2

The Java compiler does not require import declarations in a Java source code file if the fully qualified class name is specified every time a class name is used in the source code. But most Java programmers consider using fully qualified names to be cumbersome, and instead prefer to use import declarations.



3.5 Instance Variables, *set* Methods and *get* Methods

- **Variables declared in the body of method**
 - Called local variables
 - Can only be used within that method
- **Variables declared in a class declaration**
 - Called fields or instance variables
 - Each object of the class has a separate instance of the variable



Outline

GradeBook.java

```
1 // Fig. 3.7: GradeBook.java
2 // GradeBook class that contains a courseName instance variable
3 // and methods to set and get its value.
4
5 public class GradeBook
6 {
7     private String courseName; // course name for this GradeBook
8
9     // method to set the course name
10    public void setCourseName( String name )
11    {
12        courseName = name; // store the course name
13    } // end method setCourseName
14
15    // method to retrieve the course name
16    public String getCourseName()
17    {
18        return courseName;
19    } // end method getCourseName
20
21    // display a welcome message to the GradeBook user
22    public void displayMessage()
23    {
24        // this statement calls getCourseName to get the
25        // name of the course this GradeBook represents
26        System.out.printf( "welcome to the grade book for\n%s!\n",
27            getCourseName() );
28    } // end method displayMessage
29
30 } // end class GradeBook
```

Instance variable courseName

set method for courseName

get method for courseName

Call get method



Access Modifiers `public` and `private`

- **private keyword**
 - Used for most instance variables
 - `private` variables and methods are accessible only to methods of the class in which they are declared
 - Declaring instance variables `private` is known as data hiding
- **Return type**
 - Indicates item returned by method
 - Declared in method header



Software Engineering Observation 3.3

Precede every field and method declaration with an access modifier. As a rule of thumb, instance variables should be declared `private` and methods should be declared `public`. (We will see that it is appropriate to declare certain methods `private`, if they will be accessed only by other methods of the class.)



Good Programming Practice 3.1

We prefer to list the fields of a class first, so that, as you read the code, you see the names and types of the variables before you see them used in the methods of the class. It is possible to list the class's fields anywhere in the class outside its method declarations, but scattering them tends to lead to hard-to-read code.



Good Programming Practice 3.2

Place a blank line between method declarations to separate the methods and enhance program readability.



GradeBookTest Class That Demonstrates Class GradeBook

- **Default initial value**
 - Provided for all fields not initialized
 - Equal to null for Strings



set and *get* methods

- **private instance variables**
 - Cannot be accessed directly by clients of the object
 - Use *set* methods to alter the value
 - Use *get* methods to retrieve the value



Outline

GradeBookTest.java

(1 of 2)

```
1 // Fig. 3.8: GradeBookTest.java
2 // Create and manipulate a GradeBook object.
3 import java.util.Scanner; // program uses Scanner
4
5 public class GradeBookTest
6 {
7     // main method begins program execution
8     public static void main( String args[] )
9     {
10         // create Scanner to obtain input from command window
11         Scanner input = new Scanner( System.in );
12
13         // create a GradeBook object and assign it to myGradeBook
14         GradeBook myGradeBook = new GradeBook();
15
16         // display initial value of courseName
17         System.out.printf( "Initial course name is: %s\n\n",
18             myGradeBook.getCourseName() );
19     }
```

Call *get* method for courseName



Outline

GradeBookTest.java

(2 of 2)

```
20 // prompt for and read course name
21 System.out.println( "Please enter the course name:" );
22 String theName = input.nextLine(); // read a line of text
23 myGradeBook.setCourseName( theName ); // s
24 System.out.println(); // outputs a blank line
25
26 // display welcome message after specifying course name
27 myGradeBook.displayMessage();
28 } // end main
29
30 } // end class GradeBookTest
```

Call *set* method for *courseName*

Call *displayMessage*

Initial course name is: null

Please enter the course name:

CS101 Introduction to Java Programming

Welcome to the grade book for
CS101 Introduction to Java Programming!



GradeBook's UML Class Diagram with an Instance Variable and *set* and *get* Methods

- **Attributes**
 - Listed in middle compartment
 - Attribute name followed by colon followed by attribute type
- **Return type of a method**
 - Indicated with a colon and return type after the parentheses after the operation name



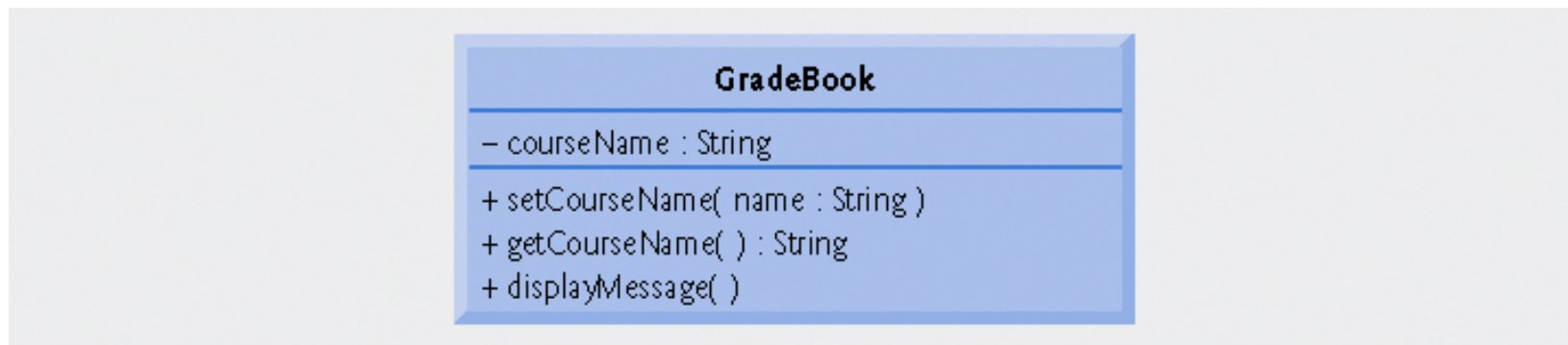


Fig. 3.9 | UML class diagram indicating that class `GradeBook` has a `courseName` attribute of UML type `String` and three operations—`setCourseName` (with a name parameter of UML type `String`), `getCourseName` (returns UML type `String`) and `displayMessage`.



Primitive Types vs. Reference Types

- **Types in Java**

- **Primitive**

- boolean, byte, char, short, int, long, float, double

- **Reference (sometimes called nonprimitive types)**

- **Objects**
 - **Default value of null**
 - **Used to invoke an object's methods**



Software Engineering Observation 3.4

A variable's declared type (e.g., `int`, `double` or `GradeBook`) indicates whether the variable is of a primitive or a reference type. If a variable's type is not one of the eight primitive types, then it is a reference type. For example, `Account account1` indicates that `account1` is a reference to an `Account` object).



3.7 Initializing Objects with Constructors

- **Constructors**
 - Initialize an object of a class
 - Java requires a constructor for every class
 - Java will provide a default no-argument constructor if none is provided
 - Called when keyword `new` is followed by the class name and parentheses



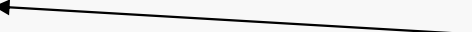
Outline

GradeBook.java

(1 of 2)

```
1 // Fig. 3.10: GradeBook.java
2 // GradeBook class with a constructor to initialize the course name.
3
4 public class GradeBook
5 {
6     private String courseName; // course name for this GradeBook
7
8     // constructor initializes courseName with String supplied as argument
9     public GradeBook( String name )
10    {
11        courseName = name; // initializes courseName
12    } // end constructor
13
14    // method to set the course name
15    public void setCourseName( String name )
16    {
17        courseName = name; // store the course name
18    } // end method setCourseName
19
20    // method to retrieve the course name
21    public String getCourseName()
22    {
23        return courseName;
24    } // end method getCourseName
```

Constructor to initialize
courseName variable



Outline

GradeBook.java

(2 of 2)

```
25
26 // display a welcome message to the GradeBook user
27 public void displayMessage()
28 {
29     // this statement calls getCourseName to get the
30     // name of the course this GradeBook represents
31     System.out.printf( "welcome to the grade book for\n%s!\n",
32         getCourseName() );
33 } // end method displayMessage
34
35 } // end class GradeBook
```



Outline

GradeBookTest.java

```
1 // Fig. 3.11: GradeBookTest.java
2 // GradeBook constructor used to specify the course name at the
3 // time each GradeBook object is created.
4
5 public class GradeBookTest
6 {
7     // main method begins program execution
8     public static void main( String args[] )
9     {
10         // create GradeBook object
11         GradeBook gradeBook1 = new GradeBook(
12             "CS101 Introduction to Java Programming" );
13         GradeBook gradeBook2 = new GradeBook(
14             "CS102 Data Structures in Java" );
15
16         // display initial value of courseName for each GradeBook
17         System.out.printf( "gradeBook1 course name is: %s\n",
18             gradeBook1.getCourseName() );
19         System.out.printf( "gradeBook2 course name is: %s\n",
20             gradeBook2.getCourseName() );
21     } // end main
22
23 } // end class GradeBookTest
```

Call constructor to create first
grade book object

Create second grade book object

```
gradeBook1 course name is: CS101 Introduction to Java Programming
gradeBook2 course name is: CS102 Data Structures in Java
```



Error-Prevention Tip 3.1

Unless default initialization of your class's instance variables is acceptable, provide a constructor to ensure that your class's instance variables are properly initialized with meaningful values when each new object of your class is created.



Adding the Constructor to Class GradeBookTest's UML Class Diagram

- **UML class diagram**
 - Constructors go in third compartment
 - Place “<<constructor>>” before constructor name
 - By convention, place constructors first in their compartment



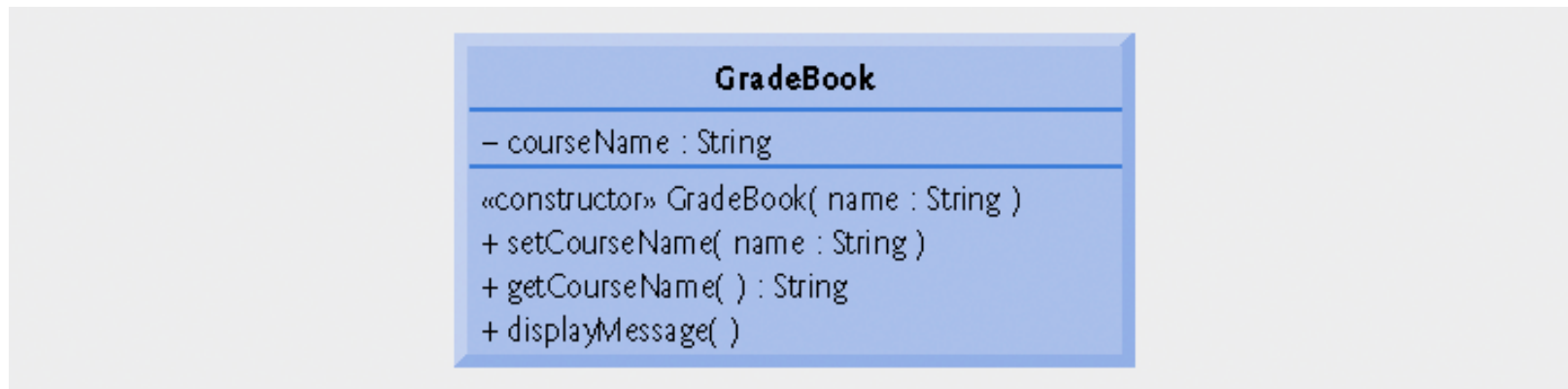


Fig. 3.12 | UML class diagram indicating that class GradeBook has a constructor that has a name parameter of UML type String.



3.8 Floating-Point Numbers and Type `double`

- **Floating-point numbers**
 - `float`
 - `double`
 - Stores numbers with greater magnitude and precision than `float`



Floating-Point Number Precision and Memory Requirements

- **float**
 - Single-precision floating-point numbers
 - Seven significant digits
- **double**
 - Double-precision floating-point numbers
 - Fifteen significant digits



Common Programming Error 3.4

Using floating-point numbers in a manner that assumes they are represented precisely can lead to logic errors.



Outline

Account.java

```
1 // Fig. 3.13: Account.java
2 // Account class with a constructor to
3 // initialize instance variable balance.
4
5 public class Account
6 {
7     private double balance; // instance variable that stores the balance
8
9     // constructor
10    public Account( double initialBalance )
11    {
12        // validate that initialBalance is greater than 0.0;
13        // if it is not, balance is initialized to the default value 0.0
14        if ( initialBalance > 0.0 )
15            balance = initialBalance;
16    } // end Account constructor
17
18    // credit (add) an amount to the account
19    public void credit( double amount )
20    {
21        balance = balance + amount; // add amount to balance
22    } // end method credit
23
24    // return the account balance
25    public double getBalance()
26    {
27        return balance; // gives the value of balance to the calling method
28    } // end method getBalance
29
30 } // end class Account
```

double variable balance

AccountTest Class to use Class Account

- **Format specifier %f**
 - Used to output floating-point numbers
 - Place a decimal and a number between the percent sign and the f to specify a precision



Outline

AccountTest.java

(1 of 3)

```
1 // Fig. 3.14: AccountTest.java
2 // Create and manipulate an Account object.
3 import java.util.Scanner;
4
5 public class AccountTest
6 {
7     // main method begins execution of Java application
8     public static void main( String args[] )
9     {
10         Account account1 = new Account( 50.00 ); // create Account object
11         Account account2 = new Account( -7.53 ); // create Account object
12
13         // display initial balance of each object
14         System.out.printf( "account1 balance: $%.2f\n",
15             account1.getBalance() );
16         System.out.printf( "account2 balance: $%.2f\n\n",
17             account2.getBalance() );
18     }
```



Outline

AccountTest.java

```
19 // create Scanner to obtain input from command window
20 Scanner input = new Scanner( System.in );
21 double depositAmount; // deposit amount read from user
22
23 System.out.print( "Enter deposit amount for account1: " ); // prompt
24 depositAmount = input.nextDouble(); // obtain user input
25 System.out.printf( "\nadding %.2f to account1 balance\n",
26     depositAmount );
27 account1.credit( depositAmount ); // add to account1 balance
28
29 // display balances
30 System.out.printf( "account1 balance: $%.2f\n",
31     account1.getBalance() );
32 System.out.printf( "account2 balance: $%.2f\n\n",
33     account2.getBalance() );
34
35 System.out.print( "Enter deposit amount for account2: " ); // prompt
36 depositAmount = input.nextDouble(); // obtain user input
37 System.out.printf( "\nadding %.2f to account2 balance\n\n",
38     depositAmount );
39 account2.credit( depositAmount ); // add to account2 balance
40
```

Input a double value

Input a double value



Outline

AccountTest.java

```
41 // display balances
42 System.out.printf( "account1 balance: $%.2f\n",
43     account1.getBalance() );
44 System.out.printf( "account2 balance: $%.2f\n",
45     account2.getBalance() );
46 } // end main
47
48 } // end class AccountTest
```

Output a double value

3)

```
account1 balance: $50.00
account2 balance: $0.00
```

```
Enter deposit amount for account1: 25.53
```

```
adding 25.53 to account1 balance
```

```
account1 balance: $75.53
account2 balance: $0.00
```

```
Enter deposit amount for account2: 123.45
```

```
adding 123.45 to account2 balance
```

```
account1 balance: $75.53
account2 balance: $123.45
```



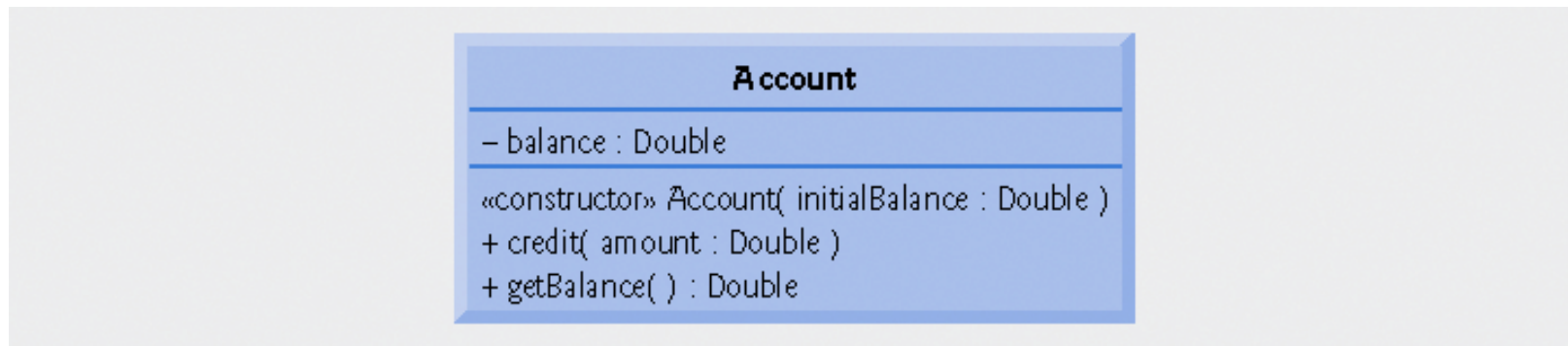


Fig. 3.15 | UML class diagram indicating that class Account has a private balance attribute of UML type Double, a constructor (with a parameter of UML type Double) and two public operations—credit (with an amount parameter of UML type Double) and getBalance (returns UML type Double).

