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.
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>3.2</td>
<td>Classes, Objects, Methods and Instance Variables</td>
</tr>
<tr>
<td>3.3</td>
<td>Declaring a Class with a Method and Instantiating an Object of a Class</td>
</tr>
<tr>
<td>3.4</td>
<td>Declaring a Method with a Parameter</td>
</tr>
<tr>
<td>3.5</td>
<td>Instance Variables, <em>set</em> Methods and <em>get</em> Methods</td>
</tr>
<tr>
<td>3.6</td>
<td>Primitive Types vs. Reference Types</td>
</tr>
<tr>
<td>3.7</td>
<td>Initializing Objects with Constructors</td>
</tr>
<tr>
<td>3.8</td>
<td>Floating-Point Numbers and Type <em>double</em></td>
</tr>
<tr>
<td>3.9</td>
<td><em>(Optional)</em> GUI and Graphics Case Study: Using Dialog Boxes</td>
</tr>
<tr>
<td>3.10</td>
<td><em>(Optional)</em> Software Engineering Case Study: Identifying the Classes in a Requirements Document</td>
</tr>
<tr>
<td>3.11</td>
<td>Wrap-Up</td>
</tr>
</tbody>
</table>
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
• 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 filename 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.
// Fig. 3.1: GradeBook.java
// Class declaration with one method.

public class GradeBook
{
    // display a welcome message to the GradeBook
    public void displayMessage()
    {
        System.out.println( "Welcome to the Grade Book!" );
    } // end method displayMessage

} // end class GradeBook
• 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
public class GradeBookTest {
    public static void main( String args[] ) {
        GradeBook myGradeBook = new GradeBook();
        myGradeBook.displayMessage();
    }
}

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
public class GradeBook {
    public void displayMessage( String courseName ) {
        System.out.printf("Welcome to the grade book for\n%s\n", courseName);
    } // end method displayMessage
} // end class GradeBook
// Fig. 3.5: GradeBookTest.java
// Create GradeBook object and pass a String to
// its displayMessage method.
import java.util.Scanner; // program uses Scanner

class GradeBookTest
{
    // main method begins program execution
    public static void main( String args[] )
    {
        // create Scanner to obtain input from command window
        Scanner input = new Scanner( System.in );

        // create a GradeBook object and assign it
        GradeBook myGradeBook = new GradeBook();

        // prompt for and input course name
        System.out.println( "Please enter the course name:" );
        String nameOfCourse = input.nextLine(); // read a line of text
        System.out.println(); // outputs a blank line

        // call myGradeBook's displayMessage method
        // and pass nameOfCourse as an argument
        myGradeBook.displayMessage( nameOfCourse );
    } // end main
} // end class GradeBookTest

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
public class GradeBook {
    private String courseName; // course name for this GradeBook

    // method to set the course name
    public void setCourseName( String name )
    {
        courseName = name; // store the course name
    } // end method setCourseName

    // method to retrieve the course name
    public String getCourseName()
    {
        return courseName;
    } // end method getCourseName

    // display a welcome message to the GradeBook user
    public void displayMessage()
    {
        // this statement calls getCourseName to get the
        // name of the course this GradeBook represents
        System.out.printf("Welcome to the grade book for\n%scourse: \
", getCourseName());
    } // end method displayMessage

} // end class GradeBook
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
// Fig. 3.8: GradeBookTest.java
// Create and manipulate a GradeBook object.
import java.util.Scanner; // program uses Scanner

public class GradeBookTest
{
    // main method begins program execution
    public static void main( String args[] )
    {
        // create Scanner to obtain input from command window
        Scanner input = new Scanner( System.in );

        // create a GradeBook object and assign it to myGradeBook
        GradeBook myGradeBook = new GradeBook();

        // display initial value of courseName
        System.out.printf( "Initial course name is: %s\n\n", myGradeBook.getCourseName() );
    }
}
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 ); // set course name
24 System.out.println(); // outputs a blank line
25 // display welcome message after specifying course name
26 myGradeBook.displayMessage();
27 } // end main
28 } // end class GradeBookTest

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
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
```java
public class GradeBook {
    private String courseName; // course name for this GradeBook

    public GradeBook( String name ) {
        courseName = name; // initializes courseName
    } // end constructor

    // method to set the course name
    public void setCourseName( String name ) {
        courseName = name; // store the course name
    } // end method setCourseName

    // method to retrieve the course name
    public String getCourseName() {
        return courseName;
    } // end method getCourseName
}
```
25 // display a welcome message to the GradeBook user
26 public void displayMessage()
27 {
28     // this statement calls getCourseName to get the
29     // name of the course this GradeBook represents
30     System.out.printf( "Welcome to the grade book for\n%s!\n",
31                         getCourseName() );
32 } // end method displayMessage
33
34 // end class GradeBook
public class GradeBookTest
{
    // main method begins program execution
    public static void main( String args[] )
    {
        // create GradeBook object
        GradeBook gradeBook1 = new GradeBook("CS101 Introduction to Java Programming");
        GradeBook gradeBook2 = new GradeBook("CS102 Data Structures in Java");
        // display initial value of courseName for each GradeBook
        System.out.printf( "gradeBook1 course name is: \n", gradeBook1.getCourseName() );
        System.out.printf( "gradeBook2 course name is: \n", gradeBook2.getCourseName() );
    } // end main
} // end class GradeBookTest

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.
public class Account
{
    private double balance; // instance variable that stores the balance

    // constructor
    public Account( double initialBalance )
    {
        // validate that initialBalance is greater than 0.0;
        // if it is not, balance is initialized to the default value 0.0
        if ( initialBalance > 0.0 )
            balance = initialBalance;
    } // end Account constructor

    // credit (add) an amount to the account
    public void credit( double amount )
    {
        balance = balance + amount; // add amount to balance
    } // end method credit

    // return the account balance
    public double getBalance()
    {
        return balance; // gives the value of balance to the calling method
    } // end method getBalance

} // end class Account
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
// Fig. 3.14: AccountTest.java
// Create and manipulate an Account object.
import java.util.Scanner;

public class AccountTest
{
    // main method begins execution of Java application
    public static void main( String args[] )
    {
        Account account1 = new Account( 50.00 ); // create Account object
        Account account2 = new Account( -7.53 ); // create Account object

        // display initial balance of each object
        System.out.printf( "account1 balance: $%.2f\n", account1.getBalance() );
        System.out.printf( "account2 balance: $%.2f\n", account2.getBalance() );
    }
}
// create Scanner to obtain input from command window
Scanner input = new Scanner(System.in);

double depositAmount; // deposit amount read from user

System.out.print( "Enter deposit amount for account1: " ); // prompt
depositAmount = input.nextDouble(); // obtain user input
System.out.printf( "adding %.2f to account1 balance\n", depositAmount );
account1.credit( depositAmount ); // add to account1 balance

// display balances
System.out.printf( "account1 balance: $%.2f\n", account1.getBalance() );
System.out.printf( "account2 balance: $%.2f\n\n", account2.getBalance() );

System.out.print( "Enter deposit amount for account2: " ); // prompt
depositAmount = input.nextDouble(); // obtain user input
System.out.printf( "adding %.2f to account2 balance\n\n", depositAmount );
account2.credit( depositAmount ); // add to account2 balance
// display balances
System.out.printf( "account1 balance: $%.2f\n",
    account1.getBalance() );
System.out.printf( "account2 balance: $%.2f\n",
    account2.getBalance() );
} // end main
}
} // end class AccountTest

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).
<table>
<thead>
<tr>
<th>Location</th>
<th>Title—Exercise(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section 3.9</td>
<td>Using Dialog Boxes—Basic input and output with dialog boxes</td>
</tr>
<tr>
<td>Section 4.14</td>
<td>Creating Simple Drawings—Displaying and drawing lines on the screen</td>
</tr>
<tr>
<td>Section 5.10</td>
<td>Drawing Rectangles and Ovals—Using shapes to represent data</td>
</tr>
<tr>
<td>Section 6.13</td>
<td>Colors and Filled Shapes—Drawing a bull’s-eye and random graphics</td>
</tr>
<tr>
<td>Section 7.13</td>
<td>Drawing Arcs—Drawing spirals with arcs</td>
</tr>
<tr>
<td>Section 8.18</td>
<td>Using Objects with Graphics—Storing shapes as objects</td>
</tr>
<tr>
<td>Section 9.8</td>
<td>Displaying Text and Images Using Labels—Providing status information</td>
</tr>
<tr>
<td>Section 10.8</td>
<td>Drawing with Polymorphism—Identifying the similarities between shapes</td>
</tr>
<tr>
<td>Exercise 11.18</td>
<td>Expanding the Interface—Using GUI components and event handling</td>
</tr>
<tr>
<td>Exercise 12.12</td>
<td>Adding Java 2D—Using the Java 2D API to enhance drawings</td>
</tr>
</tbody>
</table>

**Fig. 3.16** | Summary of the GUI and Graphics Case Study in each chapter.
Displaying Text in a Dialog Box

• Windows and dialog boxes
  – Many Java applications use these to display output
  – JOptionPane provides prepackaged dialog boxes called message dialogs
// Fig. 3.17: Dialog1.java
// Printing multiple lines in dialog box.
import javax.swing.JOptionPane; // import class JOptionPane

public class Dialog1
{
    public static void main( String args[] )
    {
        // display a dialog with the message
        JOptionPane.showMessageDialog( null, "Welcome
to
Java" );
    } // end main
} // end class Dialog1
Displaying Text in a Dialog Box

• **Package `javax.swing`**
  - Contains classes to help create graphical user interfaces (GUIs)
  - **Contains class `JOptionPane`**
    • Declares `static` method `showMessageDialog` for displaying a message dialog
Entering Text in a Dialog Box

• **Input dialog**
  – Allows user to input information
  – Created using method `showInputDialog` from class `JOptionPane`
// Fig. 3.18: NameDialog.java
// Basic input with a dialog box.
import javax.swing.JOptionPane;

public class NameDialog {
    public static void main(String args[]) {
        // prompt user to enter name
        String name = JOptionPane.showInputDialog( "What is your name?" );

        // create the message
        String message = String.format( "Welcome, %s, to Java Programming!", name );

        // display the message to welcome the user by name
        JOptionPane.showMessageDialog( null, message );
    } // end main
} // end class NameDialog
3.10 (Optional) Software Engineering Case Study: Identifying the Classes in a Requirements Document

• Begin designing the ATM system
  – Analyze the nouns and noun phrases
  – Introduce UML class diagrams
Identifying the Classes in a System

- Key nouns and noun phrases in requirements document
  - Some are attributes of other classes
  - Some do not correspond to parts of the system
  - Some are classes
    - To be represented by UML class diagrams
### Nouns and noun phrases in the requirements document

<table>
<thead>
<tr>
<th>bank</th>
<th>money / funds</th>
<th>account number</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATM</td>
<td>screen</td>
<td>PIN</td>
</tr>
<tr>
<td>user</td>
<td>keypad</td>
<td>bank database</td>
</tr>
<tr>
<td>customer</td>
<td>cash dispenser</td>
<td>balance inquiry</td>
</tr>
<tr>
<td>transaction</td>
<td>$20 bill / cash</td>
<td>withdrawal</td>
</tr>
<tr>
<td>account</td>
<td>deposit slot</td>
<td>deposit</td>
</tr>
<tr>
<td>balance</td>
<td>deposit envelope</td>
<td></td>
</tr>
</tbody>
</table>

**Fig. 3.19** | Nouns and noun phrases in the requirements document.
Modeling Classes

• 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
Fig. 3.20 | Representing a class in the UML using a class diagram.
Modeling Classes

• UML class diagrams
  – Allows suppression of class attributes and operations
    • Called an elided diagram
  – Solid line that connects two classes represents an association
    • numbers near end of each line are multiplicity values
Fig. 3.21 | Class diagram showing an association among classes.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>1</td>
<td>One</td>
</tr>
<tr>
<td>$m$</td>
<td>An integer value</td>
</tr>
<tr>
<td>0..1</td>
<td>Zero or one</td>
</tr>
<tr>
<td>$m, n$</td>
<td>$m$ or $n$</td>
</tr>
<tr>
<td>$m..n$</td>
<td>At least $m$, but not more than $n$</td>
</tr>
<tr>
<td>$*$</td>
<td>Any non-negative integer (zero or more)</td>
</tr>
<tr>
<td>0..*</td>
<td>Zero or more (identical to $*$)</td>
</tr>
<tr>
<td>1..*</td>
<td>One or more</td>
</tr>
</tbody>
</table>

**Fig. 3.22** | **Multiplicity types.**
Modeling Classes

• UML class diagrams
  – Solid diamonds attached to association lines indicate a composition relationship
  – Hollow diamonds indicate aggregation – a weaker form of composition
Fig. 3.23 | Class diagram showing composition relationships.
Fig. 3.24 | Class diagram for the ATM system model.
Fig. 3.25 | Class diagram showing composition relationships of a class Car.
Fig. 3.26 | Class diagram for the ATM system model including class Deposit.