3

Introduction to Classes and Objects (Part II)

Last time

- review: arithmetic and relational operators
- classes and objects in Java
- instance variables
- instance methods
- getters and setters

Objectives

- OOP and UML diagrams
- constructors, getters, setters, etc.
- garbage collection (GC)
- practice

Outline

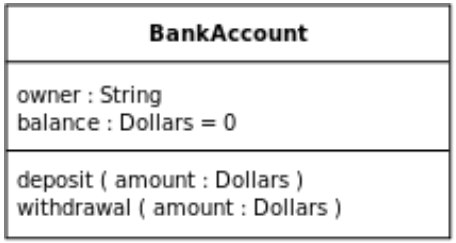
Java and OOP

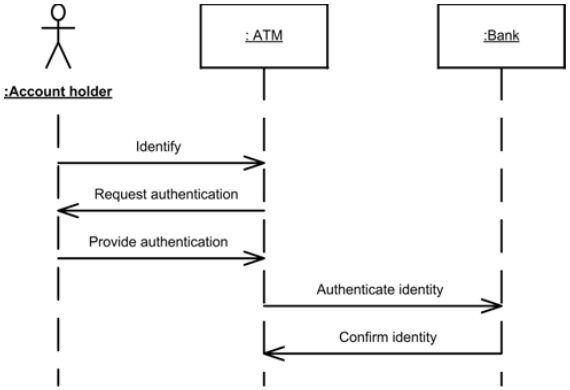
- Object-oriented programming (OOP)
- Every piece of Java code is part of a class
- Every Java class is part of a package
- UML diagrams for OOP design

Unified Modeling Language (UML)

- **Modeling language for software engineering**
- **Standardized way to visualize the design of a system**

- **Several types of diagrams:**
  - **Structural diagrams (class, package, deployment, *etc*…)**
  - **Behavioral diagram (state, sequence, communication, *etc*…)**





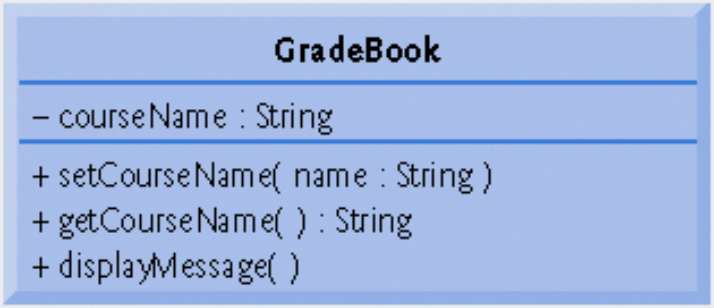


Fig. 3.9 | UML class diagram indicating that class `GradeBook` has a `courseName` attribute of UML type `String` and three operations— `setCourseName` (with a `name` parameter of UML type `String`), `getCourseName` (returns UML type `String`) and `displayMessage`.

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**

Outline

GradeBook.java

```java
// Fig. 3.7: GradeBook.java
// GradeBook class that contains a courseName instance variable
// and methods to set and get its value.

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%s!\n",
         getCourseName() );
   } // end method displayMessage

} // end class GradeBook
```

Instance variable `courseName`

set method for `courseName`

get method for `courseName`

Call *get* method

UML Example: Cube

Initializing Objects with Constructors

- **Constructors**
  - **Initialize an object of a class**
  - **Called when keyword `new` is followed by the class name and parentheses**
  - **Declared with class name followed by parentheses and input arguments**
  - **Example:**

```
public Cube()
public Cube(double sideValue)
```

Default and No-Argument Constructors

- **Every class must have at least one constructor**
  - **If no constructors are declared, the compiler will create a default constructor**
    - **Takes no arguments and initializes instance variables to their initial values specified in declaration or to default values**
  - **If constructors are declared, the default initialization for objects of the class will be performed by a no-argument constructor (if one is declared)**

Outline

GradeBook.java

(1 of 2)

```java
// Fig. 3.10: GradeBook.java
// GradeBook class with a constructor to initialize the course name.

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

   // constructor initializes courseName with String supplied as argument
   public GradeBook( String name )
   {
      courseName = name; // initializes courseNam
   } // 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
```

Constructor to initialize `courseName` variable

Outline

```
   // 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%s!\n",
         getCourseName() );
   } // end method displayMessage

} // end class GradeBook
```

GradeBook.java

(2 of 2)

Outline

GradeBookTest.java

```java
// Fig. 3.11: GradeBookTest.java
// GradeBook constructor used to specify the course name at the
// time each GradeBook object is created.

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: %s\n",
         gradeBook1.getCourseName() );
      System.out.printf( "gradeBook2 course name is: %s\n",
         gradeBook2.getCourseName() );
   } // end main

} // 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
```

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**

Local and Instance Variables

- **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 separate instance of the variable**

Default Instance Variable Values

- **Default initial value for instance variables**
  - **Provided for all fields not initialized**
  - **Equal to `null` for `String`s**
- **Local variables should be initialized**

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)

```java
// 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() );

```

Call *get* method for `courseName`

Outline

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

      // display welcome message after specifying course name
      myGradeBook.displayMessage();
   } // end main

} // 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!
```

GradeBookTest.java

(2 of 2)

Notes on *Set* and *Get* Methods

- ***Set*** **methods**
  - **Also known as *mutator* methods**
  - **Assign values to instance variables**
  - **Should validate new values for instance variables**
    - **Can return a value to indicate invalid data**

- ***Get*** **methods**
  - **Also known as *accessor* methods or query methods**
  - **Obtain the values of instance variables**
  - **Can control the format of the data it returns**

Notes on *Set* and *Get* Methods (Cont.)

- **Predicate methods**
  - **Test whether a certain condition on the object is true or false and returns the result**
  - **Example: an `isEmpty` method for a container class (a class capable of holding many objects)**

Garbage Collection (GC)

- **Garbage collection**
  - **JVM marks an object for garbage collection when there are no more references to that object**
  - **JVM's garbage collector will retrieve those objects memory so it can be used for other objects**
  - **No manual memory cleanup as in C / C++**

Method `finalize`

- `void finalize( )`
  - All classes in Java have the `finalize` method
    - Inherited from the `Object` class
  - `finalize` is called by the garbage collector when it performs termination housekeeping
  - `finalize` takes no parameters and has return type `void`