INSTANCE METHODS AND FIELDS

zyBook 9.2, zyBook 9.3, zyBook 9.4, zyBook 9.10
Oracle - Java Tutorial: Controlling Access to Members of a Class
Oracle - Java Tutorial: Using the this Keyword
Oracle - Java Tutorial: Classes

@ Dr. Jessica Young Schmidt and NCSU Computer Science Faculty

RECAP – OBJECTS

Object → A programming entity that contains state (data) and behavior (methods)

- State
  - A set of values (internal data) stored in an object.
  - Represented by fields within the class.
- Behavior
  - A set of actions an object can perform, often reporting or modifying its internal state.
  - Represented by instance methods within the class.

An object is defined by a class.

Book Object

Book object would hold data about itself (state):

Type	Field Name	Description
`String`	`title`	Name of the book
`String`	`author`	Author of the book
`int`	`pubYear`	Publication year of the book
`String`	`checkedOut`	Person who has checked out the book
`int`	`location`	Location of the book in the stacks
`Date`	`dueDate`	Date a checked out book is due

Each Book object should be able to (behavior):

Return Type	Method Name	Description
`boolean`	`checkOut(unityId)`	Check out a book
`void`	`checkIn()`	Check in a book
`int`	`getLocation()`	Info on where the book is stored
`void`	`setLocation(loc)`	Put book at this location in the library

`Date` : `https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Date.html`

SYNTAX FOR CLASS: DEFINE NEW DATA/OBJECT TYPE

Syntax for <ClassName>.java

```
public class <ClassName > {

}
```

Example

Book for Book.java

```
public class Book {

}
```

ACCESS MODIFIERS

- public
  - Visible to all classes
- private
  - Visible only to one class (this class)
- protected
  - Visible to this class and all of its subclasses, as well as to all classes in the same package as this class.
- default
  - Visible to all classes in the same package as this class.

ENCAPSULATION

- What is Encapsulation?
  - Hiding the implementation details of an object from the clients of the object
- Why Use Encapsulation?
  - Protects data from unwanted access
  - Clients cannot directly access of modify its internal workings – nor do they need to do so
  - Encapsulation leads to abstraction
  - Can later change the internal workings of the class without modifying client code
- How to Encapsulate?
  - Private fields → Private fields are visible inside the class but are not visible outside the class
  - Accessor and Mutator Methods
    - Write accessor and mutator methods to access and modify the private fields of a class
    - Maintain encapsulation because class controls the access to internal data

FIELDS/INSTANCE VARIABLES

Field → a variable inside an object that makes up part of its internal state

- Declaring field is same as normal variables: type and name
- Fields are given default initial values when an object is constructed
- Define the fields at the top of the class definition
- Difference
  - Directly inside braces of class
  - We are saying – we want every object of the class to have the variables
- Each object of type will have the fields!

```
import java.util.Date;

/**
 * An class representing a book
 *
 * @author Jessica Young Schmidt
 */
public class Book {
   /** Title of the book */
   private String title;
   /** Author of the book */
   private String author;
   /** Publication year */
   private int pubYear;
   /** Person who has checked out */
   private String checkedOut;
   /** Location of the book */
   private int location;
   /** Date book is due */
   private Date dueDate;
}
```

INSTANCE METHODS

Instance method → a method inside an object that operates on that object

- Instance methods are called on a specific object
- Instance methods can use the object's fields (object's fields have scope in the entire class)
- Syntax

```
public <type> <name>(<type><name>, ...){
                    <statement>;
                    ...
}
```

- For example, checking book in or out of the library:
  - A Book can be checked out of the library.
  - If the Book is available, we want to retrieve it from the stacks and check out the Book so that others know the Book is not available
  - Upon return to the library, a Book has to be checked in.

METHODS: STATIC VS. INSTANCE

- Static Method → A block of Java statements that is given a name.
  - Procedural Decomposition
  - If we create a static method in another client program for checking out Books, the method does not match how we think about objects: `checkOut(book1, "jdoe5");`
- Instance Method → A method inside an object that operates on that object.
  - Object-Oriented
  - We want to tie the behavior to the object itself. A Book will know how to check itself out: `book1.checkOut("jdoe5");`

BOOK EXAMPLE

- Add a checkOut instance method
  - returns true if book is not already checked out
  - checks it out to the unity id passed as parameter
  - sets the due date
  - returns false if a book is already checked out
- Add checkIn instance method
  - sets checkedOut and dueDate to null

```
import java.util.Date;

public class Book {
  /** Constant - number of milliseconds in a day */
  private static final long ONE_DAY = 86400000;
  /** Standard length of checkout in days */
  private static final int CHECKOUT_LENGTH = 90;

  // Fields would be listed here

  /**
   * Checks out a book if not already checked out
   *
   * @param unityID unityID to check book out to
   * @return true if a book if not already checked
   *   out and it checks it out to the passed in
   *   person's unity id. false if a book is
   *   already checked out
   */
  public boolean checkOut(String unityID) {
    if (checkedOut == null) {
      checkedOut = unityID;
      dueDate = new Date(System.currentTimeMillis()
        + CHECKOUT_LENGTH * ONE_DAY);
      return true;
    }
  return false;
  }

  /**
   * Checks in a book
   */
  public void checkIn() {
    checkedOut = null;
    dueDate = null;
  }

}
```

METHOD TYPES

- Mutator → an instance method that modifies the object's internal state
  - Setter method
- Accessor → an instance method that provides information about the state of an object without modifying it
  - Getter method
  - Could return the field or information about the field

```
import java.util.Date;

/**
 * An class representing a book
 * @author Jessica Young Schmidt
 */
public class Book {
    // Constants

    /** Title of the book */
    private String title;

    // Other fields

    /**
     * Returns the book title
     * @return book title
     */
    public String getTitle() {
        return title;
    }
    /**
     * Sets book title to the given parameter
     * @param title new book title
     */
    public void setTitle(String title) {
        this.title = title;
    }

    // Other methods
}
```

IMPLICIT PARAMETER

- Implicit Parameter → The object that is referenced during an instance method call.
  - Code in the instance method has implied knowledge about which object the methods is operating on
  - Instance methods ONLY execute in the context of a specific object
    b.setLocation(3);
  - We can access the implicit parameter using the keyword this. Calling a field, like author, is compiled to this.author.

```
import java.util.Date;

/**
 * An class representing a book
 * @author Jessica Young Schmidt
 */
public class Book {
    // Constants

    /** Title of the book */
    private String title;

    // Other fields

    /**
     * Returns the book title
     * @return book title
     */
    public String getTitle() {
        return title;
    }
    /**
     * Sets book title to the given parameter
     * @param title new book title
     */
    public void setTitle(String title) {
        this.title = title;
    }

    // Other methods
}
```

DEFAULT CONSTRUCTOR

- Constructor → A special method that initialized the state of new objects as they are created.
- If no constructors are included in the class, the default constructor exists that takes no parameters. It will initialize fields to their default values.
- Syntax

```
<ClassName> a = new <ClassName>();
```

- Example constructing a Book:

```
Book b = new Book();
```