CSCE 156– Lab04: Classes& Constructors

Handout

0. Prior to the Laboratory

1. Review the laboratory handout
2. Read Object Creation tutorial:
   http://download.oracle.com/javase/tutorial/java/javaOO/objectcreation.html
3. Read constructor tutorial:
   http://download.oracle.com/javase/tutorial/java/javaOO/arguments.html

1. Lab Objectives

Following the lab, you should be able to:

- Use Classes and objects to write Java programs
- Understand and use constructors
- Understand the visibility of a class's methods and how to use them

2. Lab Topics

- Classes
- Constructors, Getter/Setter Methods
- Method and variable visibility

3. Problem Statement

Java is a class-based Object Oriented Programming Language meaning that it realizes the concept of objects by allowing you to define classes, which have member methods and variables. Instances of classes are *instantiated* through a constructor; a method with the same name as a class and called using the new keyword. This lab will familiarize you with how classes and their constructors are defined and used. In addition, you will be introduced to some ways that Java supports other Object Oriented Principles: Encapsulation and Abstraction. Recall:

- **Encapsulation** is the means by which objects group data and the methods/functions that act on that data. It is also the means by which data and its methods are protected from the outside world (outside of the object). Java achieves this by allowing you to define member methods and variables and to specify the *visibility* of these fields (using the keywords `private`, `protected`, and `public`).
- **Abstraction** refers to the means by which an object exposes a public interface to the outside world while hiding the inner workings (the internal representation or the implementation

details). Java's main mechanism for supporting this is the same as with encapsulation though it does provide other means (interfaces, abstract classes).

- **Class Signaling** refers to invoking methods on an instance of a class. Java uses the dot (or period) operator to signal a class.

Importing Your Project

A Java project has been created to simulate a Library Collections system. It has several classes already defined to model Authors, Books, a Library (a collection of books) and a text-based interface which allows you to search the collection, add books to the collection, and list the collection.

Import this project into your Eclipse IDE by following either of the steps below:

From the GIT repository:

1. Go to File-> Import-> Git-> Projects from Git.
   (Note: If the "Git" import source is not listed, you might have to install a Git plugin for Eclipse)
2. Select "Clone URI" and enter the repository URI as the following
   https://git.unl.edu/csce-156/Lab04.git
3. Leave the authentication fields blank.
4. Select a branch (usually *master)* to clone from and click Next.
5. Choose a destination directory (local repository) to clone to, it must be empty. It is recommended that you choose a location on your *Z:*\. Click Next to continue.
6. Choose "Import existing Eclipse project" and click Next. Provide an appropriate project name and click Finish.
7. Note: You can commit any changes you make to the local repository, but you will not be able to push to the above URL.

Subsequent labs will provide similar Java projects. Refer back to this lab (or the "Using Eclipse and Git for labs" document in Canvas) for these instructions if you need to.

Note a few of the design elements of this project: data and methods relevant to that data are encapsulated in the relevant classes. For example, the Library class is responsible for methods acting on that library while the interface is responsible only for interface functionality.

Activity 1: Constructors

Instructions

1. Run the library program to familiarize yourself with its functionality. Note that printing the collection is not fully operational
2. Complete each of the accessor (getter) methods in the Book class; Good Practice Tip: always use `this` keyword to disambiguate the scope of variables and prevent potential problems when subclassing.
3. Observe that the Book class does not have a constructor defined. Examine the addBook code and determine how it is possible to create instances of the Book class without a constructor.

4. Modify the Book class by adding and implementing the following constructor:
   `public Book(String title, Author author, String isbn, String publishDate) { ... }`
5. Adding this constructor will cause syntax errors in other parts of the program. Think about why and then fix these problems by modifying the code appropriately.

Activity 2: Enforcing Good Encapsulation

The Book class is well designed: it logically groups data and methods together that semantically define what a book is and how you can use it. The `Author` class is not well defined though; its data members are publicly exposed and it has no methods at all.

Instructions

1. Redesign the `Author` class and make its member variables private.
2. Create and use getter methods to make the members accessible to the outside world. Use these methods where appropriate.
3. Create setter methods (called mutator methods) to enable code outside of the `Author` class to change the member variables. Add some data validation: for example, do not allow "invalid" values for member variables.
4. Add and make use of an appropriate constructor to this class.
5. Add a method to return a `String` that is the author's last name/first name separated by a comma and then utilize it where appropriate (modify the `printBooks` method to use this new method instead of formatting the last name/first name directly).

Activity 3: Adding and Using Methods

The `printBooks` method prints out the title, author, and ISBN in a formatted manner one per line. In this activity, you will modify it to output additional information: the year of its publication and its age (the number of years since its publication date).

Instructions

1. Modify the `printBooks` method in the `LibraryDemo` class to output the publication year of each book in another column. Note that the `Book` class offers a `getPublishDate()` method already implemented for you.
2. To add the "age" column, you will need to add a new method to the Book class that returns the number of years between the publish date and today. You may find the following code snippet useful (it utilizes the Joda Time library already imported into the project):
   `int years = new Period(this.publishDate, DateTime.now()).getYears();`

   The class `Period` is also present in `java.util` package. Make sure to import the one from the joda time library (`org.joda.time.Period`).

Advanced Activities (Optional)

1. The `printBooks` method prints books in the order in which they appear in the list. This isn't as useful as if they were sorted in some manner. Read Oracle's tutorial on Object ordering,

http://docs.oracle.com/javase/tutorial/collections/interfaces/order.html
Write code to use the `Collections.sort()` method along with an anonymous `Comparator<Book>` instance to sort the collection according to the book's title.

2. Composition is when classes own instances of other classes (the Book class owns an instance of the Author class). If different instances of a class A_1 and A_2 are given references to the same object *B*, then changes are made to *B*, the changes will be apparent to both A_1 and A_2. Sometimes this behavior is desired. Other times it is not. One common method to prevent this is to define a *copy constructor*, which takes another instance of the class and performs a *deep copy* of an object by copying each of its fields to the new instance. For example, the copy constructor for the Author class may look something like this:

```
public Author(Author author) {
  this.firstName = new String(author.firstName);
  this.lastName = new String(author.lastName);
}
```

Design and implement a copy constructor for the Book class and the Library class. How do different fields need to be copied?