CSCE 156 – Computer Science II

Lab 4.0 - Classes & Constructors

Prior to Lab

1. Review this laboratory handout prior to lab.
2. Read constructor tutorial:
   `https://docs.oracle.com/javase/tutorial/java/javaOO/constructors.html`
3. Read Object Creation tutorial:
   `http://download.oracle.com/javase/tutorial/java/javaOO/objectcreation.html`

Lab Objectives & Topics

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

Peer Programming Pair-Up

To encourage collaboration and a team environment, labs will be structured in a *pair programming* setup. At the start of each lab, you will be randomly paired up with another student (conflicts such as absences will be dealt with by the lab instructor). One of you will be designated the *driver* and the other the *navigator*.

The navigator will be responsible for reading the instructions and telling the driver what to do next. The driver will be in charge of the keyboard and workstation. Both driver and navigator are responsible for suggesting fixes and solutions together. Neither the navigator nor the driver is "in charge." Beyond your immediate pairing, you are encouraged to help and interact and with other pairs in the lab.

Each week you should alternate: if you were a driver last week, be a navigator next, etc. Resolve any issues (you were both drivers last week) within your pair. Ask the lab instructor to resolve issues only when you cannot come to a consensus.

Because of the peer programming setup of labs, it is absolutely essential that you complete any pre-lab activities and familiarize yourself with the handouts prior to coming to lab. Failure to do so will negatively impact your ability to collaborate and work with others which may mean that you will not be able to complete the lab.

Getting Started

Clone the project code for this lab from GitHub in Eclipse using the URL, `https://github.com/cbourke/CSCE156-Lab04`. Refer to Lab 01 for instructions on how to clone a project from GitHub.

Classes & Constructors

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.

- *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 a class's data is 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. That is, to invoke one of its methods.

Activities

We have provided a Java project that simulates 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.

Constructing Constructors

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. Best Practice Tip: always use the `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.

```java
public Book(String title, Author author, String isbn,
                 String publishDate) {
   //TODO: your code here
}
```

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.

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.

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 (also 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).

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).

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 date/time library provided by Java 8):

   ```
   int years = Period.between(this.publishDate, LocalDate.now()).getYears();
   ```

Advanced Activity (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 a class owns instance(s) 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 an 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:

```java
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?