L01D: Assignment - Arrays

Submit Assignment

Due Sunday by 11:59pm **Points** 10 **Submitting** a file upload **File Types** zip
Available until Sep 20 at 11:59pm

After reading all **L01C** content pages in Lesson 01C, you will complete this assignment according to the information below.

Do not use the scanner class or any other user input request. You application should be self-contained and run without user input.

Assignment Objectives

1. Practice on creating and using Arrays as attributes and as input parameters. Keep practicing MVC.

Deliverables

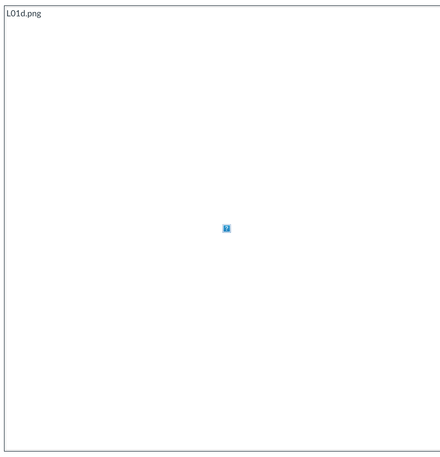
A zipped Java project according to the How to submit Labs and Assignments guide.

O.O. Requirements (these items will be part of your grade)

1. One class, one file. Don't create multiple classes in the same .java file
2. Don't use static variables and methods
3. Encapsulation: make sure you protect your class variables and provide access to them through get and set methods
4. all the classes are required to have a constructor that receives **all** the attributes as parameters and update the attributes accordingly
5. Follow Horstmann's Java Language Coding Guidelines
6. Organized in packages (MVC - Model - View Controller)

Contents

L01d.png



Data

- The Model class will create 5 Person objects (same as last assignment)
  - using the full-parameter constructor with the data below
    - name=Marcus Allen, height=5'2", weight=200, hometown=Upper Marlboro, Md., highSchool=Dr. Henry A. Wise, Jr.
    - name=Kyle Alston, height=5'9", weight=180, hometown=Robbinsville, N.J., highSchool=Robbinsville
    - name=Troy Apke, height=6'1", weight=220, hometown=Mt. Lebanon, Pa., highSchool=Mount Lebanon
    - name=Matthew Baney, height=6'0", weight=225, hometown=State College, Pa., highSchool=State College
  - using the no-parameter constructor

Functionality

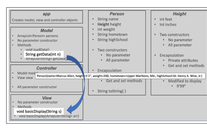
This assignment is a follow up from the previous assignment with a major update:

- Model will store the Person objects in an ArrayList instead of using individual variables.
  - ArrayList<Person> persons is **the only attribute** in Model
- You will also use the concept of method **overloading**, which you have seen in the beginning of Chapter 7.
  - from Schildt's "Java: The Complete Reference, Eleventh Edition, 11th Edition":
    - *"In Java, it is possible to define two or more methods within the same class that share the same name, as long as their parameter declarations are different. When this is the case, the methods are said to be overloaded, and the process is referred to as method overloading. "*

You will keep the MVC functionality from the previous assignment

- Controller will be simpler having only one line calling the new method in Model, ArrayList<String> getData(). This method returns an ArrayList<String> with one line (one big String) for each Person in the ArrayList.
- Model will need to have an updated *loadData( )* because the *Person* objects are now stored in an ArrayList.
- Model will also **overload getData**
  - public String getData(int n)
  - public ArrayList<String> getData()
- View will **overload basicDisplay**
  - public void basicDisplay(String s)
  - public void basicDisplay(ArrayList<String> arr)

L01C **L01D**



As in the previous assignment, the main objective is to make View display the information about the 5 Person objects. These 5 objects are created in Model in the method loadData( ). Controller will retrieve the data from Model using the method ArrayList<String> getData() and pass it to View. View will use the method void basicDisplay(ArrayList<String> arr) to display data from each object with the following result:

```
Person{name=Marcus Allen, height=5'2", weight=200, hometown=Upper Marlboro, Md., highSchool=Dr. Henry A. Wise, Jr.
,}
Person{name=Kyle Alston, height=5'9", weight=180, hometown=Robbinsville, N.J., highSchool=Robbinsville}
Person{name=Troy Apke, height=6'1", weight=220, hometown=Mt. Lebanon, Pa., highSchool=Mount Lebanon}
Person{name=Matthew Baney, height=6'0", weight=225, hometown=State College, Pa., highSchool=State College}
Person{name=, height=0'0", weight=0, hometown=, highSchool=}
```

The classes

- **App**
  - it has the *main* method which is the method that Java looks for and runs to start any application
  - it creates 3 objects, one of the Model class, one of the View class and one of the Controller class.
  - When it creates Controller, it passes the two other objects as input parameters so that Controller has access to Model and View.

```
Model model = new Model();
View view = new View();
Controller controller = new Controller(model, view);
```

- **Controller**
  - will retrieve data from Model using the model object to call now the method ArrayList<String>getData ( ) (previously called the method getData (int n))
    - ArrayList<String> getData without the "int n" parameter returns all objects in the ArrayList as Strings.
  - will pass the data to View, which will display it

For instance, controller **might** look like this:

```
view.basicDisplay(model.getData());
```

- **Model**
  - it has only one attribute, ArrayList<Person> persons
  - it uses encapsulation
  - it needs an updated method, *loadData( )*, to load the data, now using an ArrayList to store the 5 Person objects
  - it needs a method, *ArrayList<String> getData()*, which **overloads** String getData(int n) and returns an ArrayList of String with one big String for each Person object in the array
- **View**
  - it just needs to display the text data that it will receive from Controller
  - It needs an overloaded method, void *basicDisplay (String s)*. The new method receives an ArrayList of Strings and displays each String in a new line, it will be void *basicDisplay(ArrayList<String> arr)*
- **Person**
  - uses encapsulation
    - private attributes
    - a get and a set method for each attribute
  - has the following attributes
    - String name;
    - **Height** height;
    - int weight;
    - String hometown;
    - String highSchool;
  - has two constructors
    - one with no parameters
    - one with all the parameters (one for each attribute)
  - a toString( ) method

Height

- it is a class (or type) which is used in Person defining the type of the attribute *height*
- uses encapsulation
  - private attributes
  - a get and a set method for each attribute
- it has two attributes
  - int feet;
  - int inches
- two constructors
  - one with no parameters
  - one with all the parameters (one for each attribute)
- and a method
  - String toString( )
    - toString( ) overrides the superclass Object toString( ) method
    - toString( ) returns information about this class attributes as a String
    - it returns a formatted String with feet and inches
      - for instance: 5'2"

Output

The output will look like this:

```
Person{name=Marcus Allen, height=5'2", weight=200, hometown=Upper Marlboro, Md., highSchool=Dr. Henry A. Wise, Jr
,}
Person{name=Kyle Alston, height=5'9", weight=180, hometown=Robbinsville, N.J., highSchool=Robbinsville}
Person{name=Troy Apke, height=6'1", weight=220, hometown=Mt. Lebanon, Pa., highSchool=Mount Lebanon}
Person{name=Matthew Baney, height=6'0", weight=225, hometown=State College, Pa., highSchool=State College}
Person{name=, height=0'0", weight=0, hometown=, highSchool=}
```