Introduction to Java

CS9053

Tuesday 6 PM – 8:30 PM

Prof. Dean Christakos

Sept. 23, 2020

Due: Sept. 29, 2020

**Assignment 3**

Part I – Arrays

1. In LargestPairProduct.java, there is a variable intArray which contains 10 integers. Find the largest product of all pairs of numbers in the array and display the result. Nested loops will probably be the right thing to do:

(get each index that will be the first member of the pair)

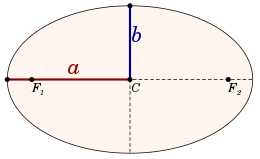
(get each index that will be the second member of the pair)

For efficiency’s sake you will want to avoid multiplying pairs repetitively. I.e., if you multiple intArray[5]\*intArray[6], there is no need to multiply intArray[6]\*intArray[5]

1. In TwoDimensionalArray.java, we have a two dimensional array called twoDimArray and two one-dimensional arrays, arrayOne and arrayTwo. You will copy the CONTENTS of arrayOne into twoDimArray[0] and the CONTENTS of arrayTwo into twoDimArray[1]. Use two nested loops and index 0 of the first dimension of twoDimArray means choosing arrayOne as the array to copy and index 1 of the first dimension means choosing arrayTwo.

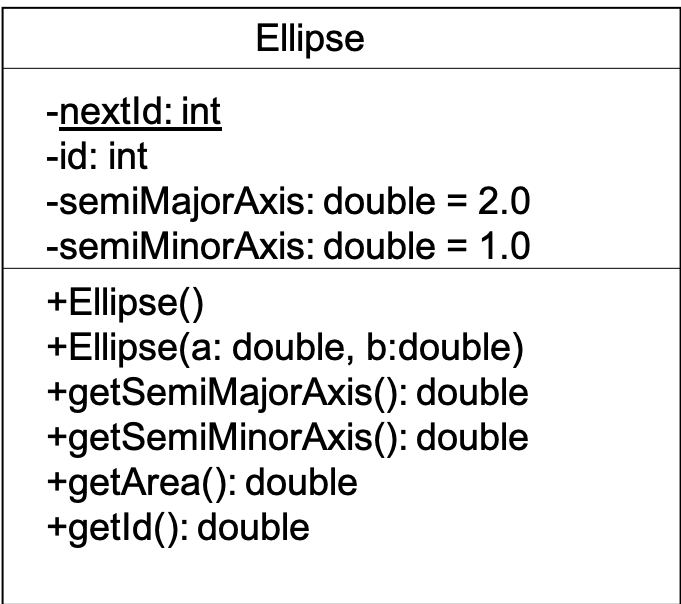
Part II – Creating objects

1. Ellipse: In the lecture you have seen the creation of a circle. Here you are going to create an Ellipse. An Ellipse is a general case of a circle, except that instead of a single radius, it has a semi-major axis (the longer axis), usually referred to as *a*, and a semi-minor axis (the shorter axis), usually referred to as *b*. *b* ***<=*** *a*.



The area of an ellipse is given by *a x b x π.*

You will create a class Ellipse using the following UML:



**In standard UML parlance, “+” indicates that a field or method is public and “-“ indicates that a field or method is private. An underlined field or method indicates it is static.**

You don’t have to do any strict enforcement of the requirement that a >= b or semiMajorAxis >= semiMinorAxis, but you should assume it works that way and any code you use should assume that.

Every time you create a new Ellipse instance, it should have a new sequential id, based on the value of nextId, which should be incremented every time you create a new Ellipse instance.

1. Holiday: An object of class Holiday represents a holiday during the year. An object of class Holiday represents a holiday during the year. This class has three instance variables:

● name, which is a String representing the name of the holiday

● day, which is an int representing the day of the month of the holiday

● month, which is a String representing the month the holiday is in

public class Holiday {

private String name;

private int day;

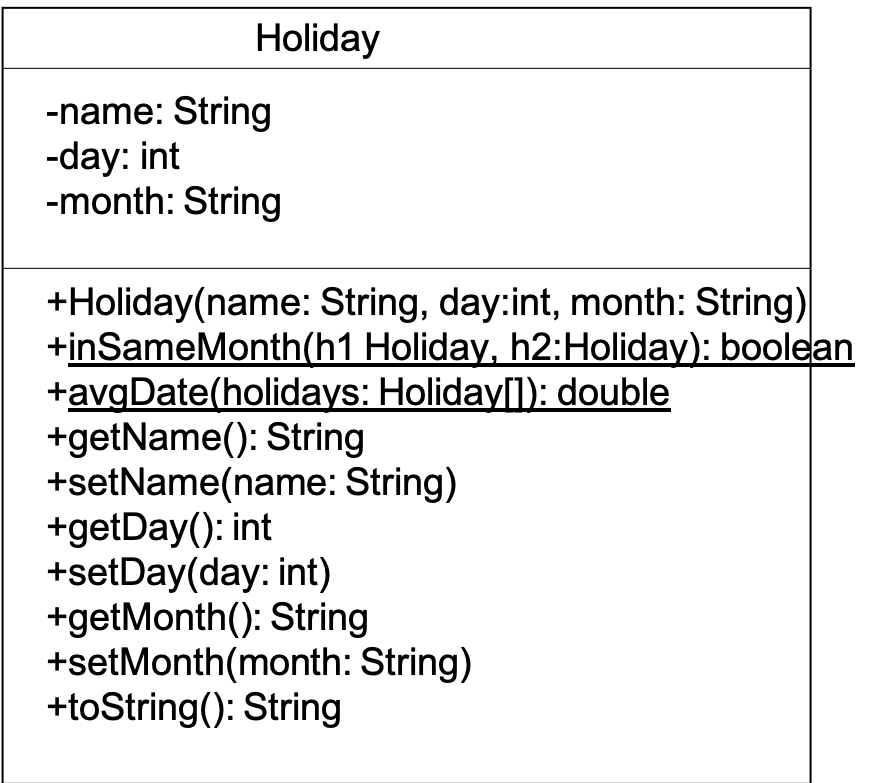
private String month;

// your code goes here

}

* 1. Write a constructor for the class Holiday, which takes a String representing the name, an int representing the day, and a String representing the month as its arguments, and sets the class variables to these values.
  2. Write a static method inSameMonth, which compares two instances of the class Holiday, and returns the Boolean value true if they have the same month, and false if they do not. Show how it is used
  3. Write a static method avgDate which takes an array of base type Holiday as its argument, and returns a double that is the average of the day variables in the Holiday instances in the array. You may assume that the array is full (i.e. does not have any null entries). Show how it is used
  4. Write getters and setters to access the name, day, and month fields
  5. Write a method toString that returns the name of the holiday followed by the date. For example, if in the Holiday object name = Mayday, month = May, and day = 1, then toString should return “Mayday: May 1”
  6. Write a piece of code that creates a Holiday instance with the name “Independence Day”, with the day “4”, and with the month “July”.

Here is the UML:



Once again, **in standard UML parlance, “+” indicates that a field or method is public and “-“ indicates that a field or method is private. An underlined field or method indicates it is static.**

I have written the code to create a few holidays and put them in an array to get you started so you can test out the methods.

**Part II: Bank Accounts**

1. Here we have two classes, a bank and an account. In the main method in Bank.java, there’s an infinite loop that lists all the accounts, prints out their balances, and prompts you to do a transfer from one account to another. By default the bank has 5 accounts in NUM\_ACCOUNTS, which are in an array of Account objects. The account, in Account.java has an account id, and a balance. There should be methods to get the id and the balance, as well as methods to withdraw money and deposit money. deposit(double amount) should add money to the balance. withdraw(double amount) should subtract money to the balance if there is enough money in the account, and return true if so. If the withdrawal exceeds the balance, the withdraw method should return false.

As with the Circle and Ellipse, every time you create an account, it should generate a new id.

I’ve written the control code in the main method in Bank.java, so you can see what it does. Your job is to fill in the code for the Bank and Account objects to make it happen.

When you create a Bank object, it should by default create 5 accounts with a balance of $1000.

In Bank.java, I have written all of the code in the main method. I have provided skeleton methods for **some** but not all of the methods and fields you must implement given the UMLs. It’s your responsibility to implement everything in the UMLs.

The UMLs are as follows. A static field in all caps indicates it is final (ie, a constant):

