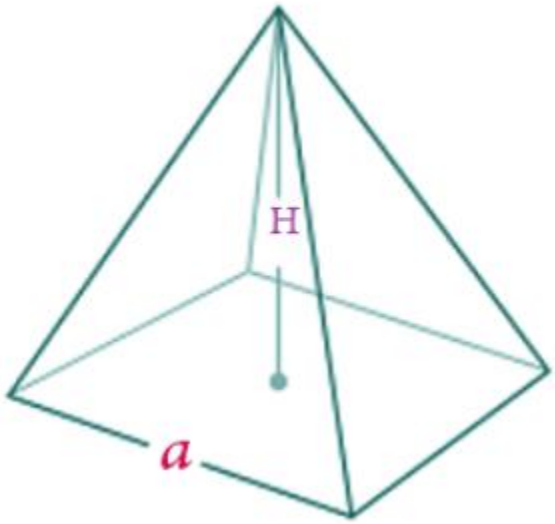
Introduction to Java Section I
CS9053
Thursday 6:00 PM – 8:30 PM
Prof. Dean Christakos
February 9th, 2023
Due: February 17th, 2023 11:59 PM

Assignment 3

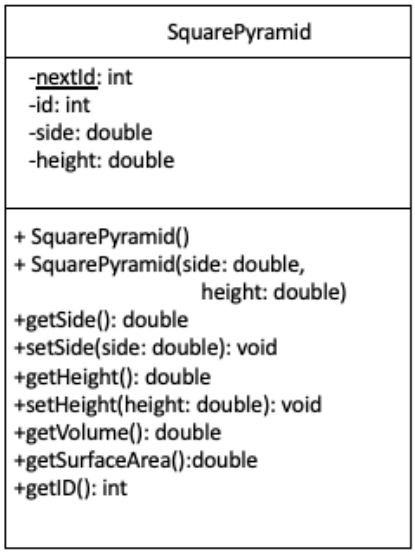
Part I – Creating objects

1. Square pyramid: In the lecture you have seen the creation of a circle. Here you are going to create a square pyramid. A square pyramid has a square base. It has base, *a*, and a height, *H*.



The Volume of a Pyramid is given by $\frac{1}{3}a^2H$
l is known at the "slant" and is given by $\sqrt{h^2 + r^2}$
The surface area of a square pyramid is given by $a^2 + 2a\sqrt{\frac{a^2}{4} + H^2}$

You will create a class SquarePyramid 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.

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

2. Dog: An object of class Dog represents a dog. This class has three instance variables:
   - `age`, which is an `int` representing the age of the dog
   - `owner`, which is a `String` representing the name of the owner
   - `breed`, which is a `String` representing breed of the dog

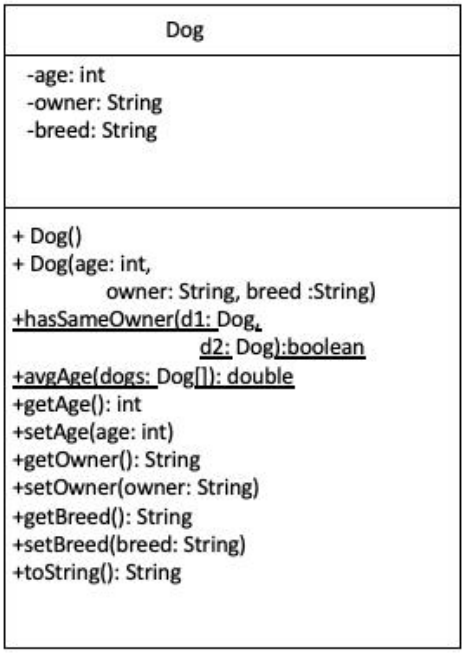
```
public class Dog {
      private int age;
      private String owner;
      private String breed;
      // your code goes here
}
```

   a. Write a constructor for the class Dog, which takes an `int` representing the age, a `String` representing the owner, and a `String` representing the breed as its arguments, and sets the class variables to these values.
   Note/Hint: you'll note in the UML below that the parameter names are the same as the fields/instance variables. To specify an instance variable in a class method, you can specify "this". For example:

```
public void myMethod(int age) {
      System.out.println(age); // prints the value of age in
the parameter
      System.out.println(this.age); // prints the value of the
instance variable/field "age"
      this.age = age; // assigns the value of the parameter age
to the instance variable.field "age"
}
```

b. Write a static method `hasSameOwner`, which compares two instances of the class Dog, and returns the boolean value **true** if they have the same owner, and **false** if they do not. Show how it is used
c. Write a static method `avgAge` which takes an array of base type Dog as its argument, and returns a double that is the average of the age variables in the Dog 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
d. For each field, write a method get**Field**() and set**Field** which will return the value of that field and which you can set the value of the field to the value you've passed
e. Write a method `toString` that returns the name of the holiday followed by the date. For example, if in the Dog object age = 12, owner = "John Smith", and breed = "Labrador", then `toString` should return a String "Labrador: Owner: John Smith, Age: 12"
f. Write a piece of code that creates a Dog instance with the owner "Dexter Morgan", with the age "8", and with the breed "Corgi".

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 Dogs 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`.

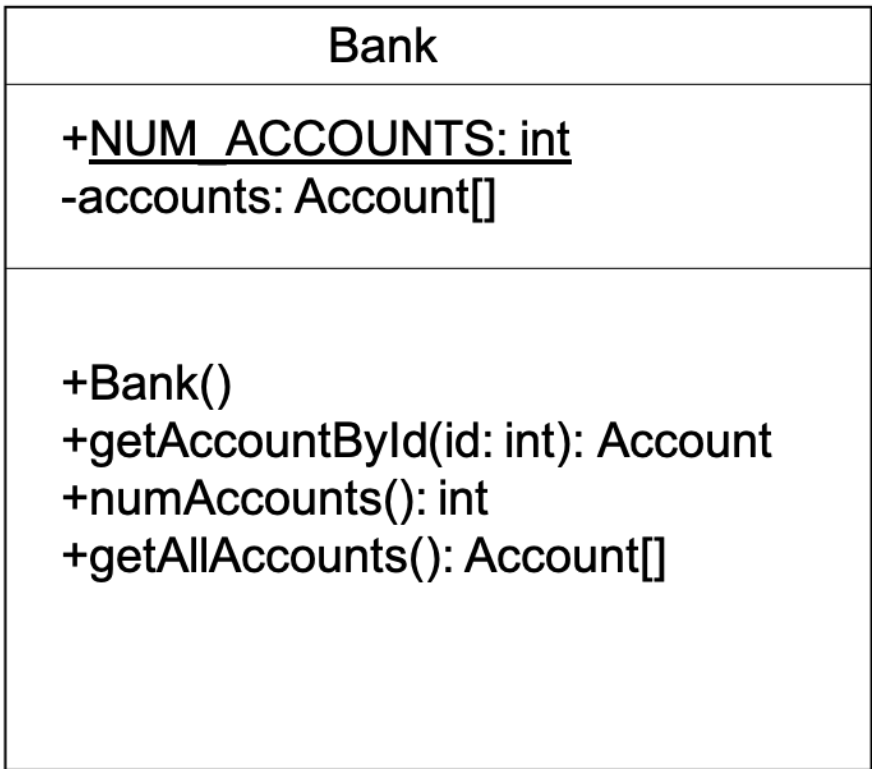
   As with the `Circle` and `SquarePyramid`, every time you create an account, it should generate a new id.

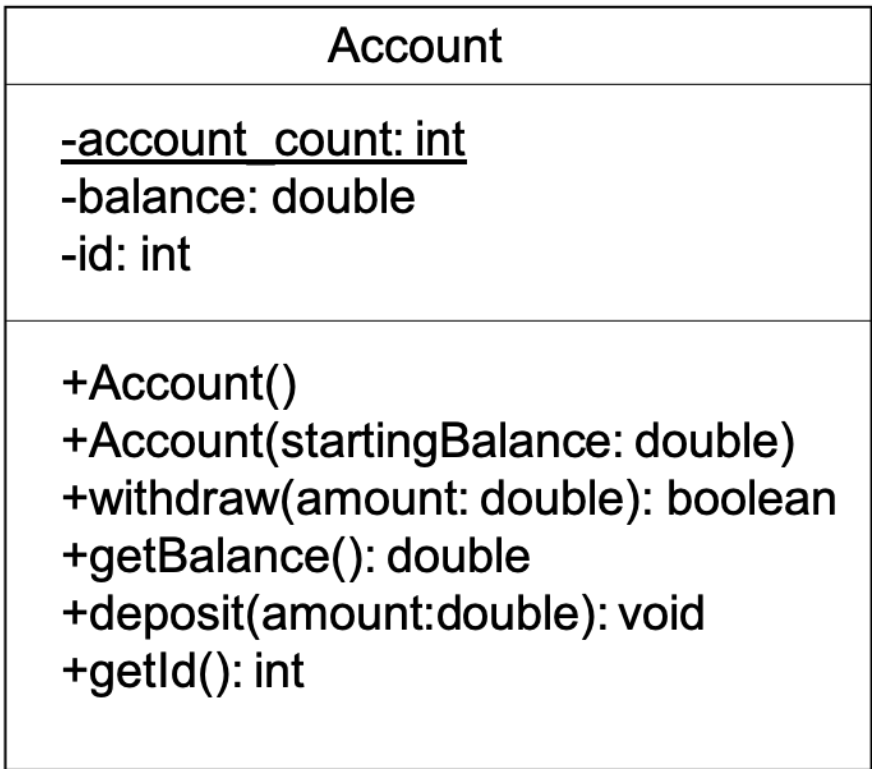
I've written the control code in the main method in `Bank.java`, so you can see what it does. Your job is to fill in the code for the `Bank` and `Account` objects to make it happen.

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):




Account
-account_count: int
-balance: double
-id: int
+Account()
+Account(startingBalance: double)
+withdraw(amount: double): boolean
+getBalance(): double
+deposit(amount:double): void
+getId(): int