Lab 17 (November 6 or November 7)

Instructions: Complete the exercises below. Be sure to show your code to one of the lab TAs before you leave, so that you can receive credit for this lab. You must also upload a copy of all your source code (`.java`) files through the link on Blackboard by 11:59 PM on Tuesday, November 7.

1. Implement a class named `StackOfObjects` that can (theoretically) store an infinite number of values. This class has the following instance variables and methods:

   * A `private Object` array. This array should be declared as an instance variable, but not instantiated until the constructor (described below) executes. As the array fills up, it will be replaced with a new, larger array that contains all of its previous contents.

   * A `private` integer variable named `size` that counts the current number of elements stored in the array. (`size` – 1) is the index of the element at the top of the stack.

   * A `public` no-argument constructor that initializes the array instance variable with an initial capacity of 2. The constructor initializes `size` to 0; the array is initially empty.

   * A `public` (no-argument) `size()` method that returns the number of stack items.

   * A `public push()` method that takes a single `Object` argument and inserts it into the array at the first available index (i.e., `size`), incrementing `size` afterward. If the array is already full before you insert the new value, call `expand()` (described below) first to resize the internal array before performing the insertion.

   * A `public pop()` method that does not take any arguments. This method removes the top element of the stack (meaning the item at index (`size` – 1)) and decrements `size` before returning the value that was removed. You may assume that the user will never attempt to pop a value from an empty stack.

   * A `private` helper method named `expand()` that takes no arguments and does not return anything. This method creates a new `Object` array that is twice as large as the current internal array, copies the old array's contents into the new array, and then assigns the new (larger) array to the class instance variable (replacing the old array). This method does not modify `size`; the new array initially has the same contents as the old array.

   Write some code (a `main()` method inside your class or in a separate driver class) to test your StackOfObjects class by repeatedly pushing and popping values and displaying the stack size as it changes. For simplicity, use `Strings` (which are objects) as your stack data.

2. Design a class named `Account` that contains:

* A `private int` data field named `id` (whose default value is 0).

* A `private double` data field named `balance` (whose default value is 1000).

* A `private double` data field named `annualInterestRate` (whose default value is 4.50).

* A no-arg constructor that creates a new `Account` object with default values.

* A second constructor that creates an `Account` with a specified id and non-negative initial balance (and the default interest rate).

* A third constructor that creates an `Account` with a specified id, non-negative initial balance, and non-negative interest rate.

* `public` accessor and mutator methods for all three instance variables.

* A `public` method named `getMonthlyInterestRate()` that returns the monthly interest rate (`annualInterestRate/12`).

* A `public` method named `withdraw()` that takes a `double` argument. If the account balance is smaller than that amount, print a statement saying that there are insufficient funds and leave the balance unchanged. Otherwise, deduct the argument value from the current account balance.

* A `public deposit()` method that adds the value of its (positive) `double` argument to the current account balance.

* A `public toString()` method that prints the values of each instance variable in an appropriate format.

Next, create a subclass of `Account` named `CheckingAccount`. A `CheckingAccount` has an overdraft limit (a `double` instance variable whose value is set by an additional constructor and a mutator method); if you attempt to withdraw more money than the account currently has, the withdrawal still succeeds as long as the total amount is less than or equal to the current balance plus the overdraft limit (the account balance just becomes negative after this). Override `toString()` to include the overdraft limit.

Write a test program that creates one `Account` and one `CheckingAccount` and invokes their `toString()` methods.

Grading Guidelines: This lab is graded on a scale of 0-3 points, assigned as follows:

0 points: Student is absent or does not appear to have completed any work for the lab

1 point: Student has written only one program, but it does not compile or run at all due to errors.

2 points: Student has written (or attempted to write) both programs, but only one compiles and runs without error.

3 points: Student has written both programs, and they both compile and run correctly, without any apparent errors.