# **CIS 225 101 – Spring 2014**

# **Project #1**

**Name:**  **Due Date** :

|  |  |
| --- | --- |
| Documentation (30):  Cover Page (2)  Table of Content (2)  Abstract (5)  Design (UML diagram and Javadocs for Customer and Bank classes) (11)  Source Code (5)  Sample Input (5) |  |
| Sample Output(10) |  |
| Correct Logic(40):  Clean compile |  |
| Structure(20):  white spacing (5)  proper indentation(5)  meaningful names(5)  logical flow(5) |  |
| Total points: out of 100% |  |

**AT THE BEGINNING OF LAB (you must demonstrate to me that the program is working)**

Your contribution to the project in percent\_\_\_\_\_\_\_\_\_\_\_\_\_

Your partner’s contribution to the project in percent\_\_\_\_\_\_\_\_\_\_\_\_\_

Explain in Detail your contribution:

**Comments:**

**Due Date** :

**LAB March 3rd – week 5 (you have to demonstrate to me that the lab is working)**

**Hand In:**

In a **closeable** large envelop (no folders) with your name clearly stated outside:

* A **labeled storage device** with the source code ( the programs with the. java file ) and the .class files.
* A **hard copy** of the source code programs stapled together.
* A **hard Copy** of the **test cases** stapled together with your name marked on top
* You will be required to show me that the programs are working

# **CIS 225 101 – Spring 2014**

# **Project #1**

* **Purpose:** Array of objects
* **Assignment:**

The USB Bank can handle up to 30 customers who have saving account and checking account.

Design and implement a program that manages the accounts.

Each customer has a name, phone number , an id, and account balance.

Allow each customer to make deposit and withdrawal. Produce appropriate error message for invalid transaction.

To solve the problem you are to create a **Customer** account, **CheckingAccount** class which is a child of Customer class and **SavingAccount** class, which is a child of Customer class. You should create also a **Bank** class which will hold the list of customers( Savings or Checking)

* Create a **Customer** class:
* With the customer name ( last name, first name), id ( a string)
* Create three contactors:
  + A default constructor that set the last name, first name, and the id to empty strings.
  + A constructor which accepts last name, first name, id.
  + A constructor that accepts only the first name and last name. ( the constructor will set the id to empty string)
  + A constructor that accepts only the id
* Create a Checking account class which is a child of Customer
* Create a Savings account class which is a child of Customer
* The Checking and Saving classes each will have:
  + Variable called balance (double) ( no other variables).
  + Three contactors:
  + A default constructor that set the last name, first name, and the id to empty strings, and the balance to zero
  + A constructor which accepts last name, first name, id, and account balance.
  + A constructor that accepts only the first name and last name. (the constructor will set the id to empty string, and the balance to zero)
  + A constructor that accepts only the id, will set the first name, last name to empty string and the balance to zero
  + Methods to
  + **Deposit** money to the account ( called Deposit)

**public void deposit(double amount);**

After the deposit provide a message:

* Display the message:

Customer: last name first name , id

Type of account ( Savings or Checking)

Balance : <put here the new balance>

* If try to deposit a negative amount, generate an error message:

---ERROR---invalid deposit amount

Customer: last name first name, id

Type of account ( Savings or Checking)

Balance: <the old balance>

Amount : <put here the amount that was trying to deposit>

* **Withdraw** from the accounts (called Withdrawal).

**public void withdraw(double amount);**

After the withdraw provide a message:

Customer: last name first name, id

Type of account ( Savings or Checking)

Balance : <put here the new balance after withdraw>

* If try to withdraw a negative number

Generate an error:

---ERROR--- invalid withdrawal amount

Customer: last name first name, id

Type of account ( Savings or Checking)

Balance ; < put here the old balance>

Amount Requested: <put here the amount that was requested>

* If try to withdraw more money then have in balance then withdraw only the balance ( i.e. set the account to zero) and give a message:

---ERROR: insufficient funds

Customer: last name first name, id

Type of account ( Savings or Checking)

Balance : <put here the new balance>

Amount Requested : put here the amount that was requested

Amount Received: put here the available balance

* **These classes ( all three) overrides the following methods:**
* **toString**  which returns the last name, first name, id, and balance, type of account ( Savings or Checking)

**public String toString()**

* **equals** to compare if two customers are the same

**public boolean equals(Object obj) (** the parameter has to be Object**)**

This method checks if two customers are equal to each other by the id, if the id is empty string check by last name and first name

* **compareTo public int compareTo(Comparable obj)**

Look at the method description in the Comparable interface. compare the objects by the id.

* Create **Bank class** that has:
* the **list** of customers as private data,
* and the **count** of actual number of customers in the bank

The bank class should have methods to:

* **Add** a new customer to the list

**public void add (Customer aCustomer)**

If the list is not full, this method will add a new customer to the end of the list.

If the list is full, the method will call a method **insureCapacity()** to increase the array size, and will add a new customer to the end of the list.

* **Increase the array size**

**private void InsureCapacity()**

This is a private method, which creates a new temporary array which has the capacity the original array plus 10 more, and stores in the array the data from the original array, and then assign **list** to the temporary array.

* **Delete** a customer from the account, if the customer is not in the list return false, else return true ( this method can call find method for help)

**public boolean delete (Customer aCustomer)**

* Find index of the customer in the list. This private method will return the index of the customer in the list, or return -1 if the customer is not in the list

**private int FindIndex (Customer aCustomer)**

* **Retrieve** a customer (will return an object of the class Customer)

**public Customer retrieve (Customer aCustomer)**

* **Sorting the array**

Sorting the array in ascending order by Last name and then First Name

**public static sort (Comparable[] list)**

***3****.CREATE a UML diagram for each class*

4. Write a test class called **BankTest** ( this class has **main** in it)

* This class has a **menu(String st) method, this method has the following options:**

1. Add a customer
2. Deposit
3. Withdraw
4. Retrieve a customers
5. Remove a customer
6. Sort the customers (in ascending order by last name and first name)
7. Print the list of customers ( last name first name, id, and balance)
8. Quit

* The main method is doing the followings:

// create an object of the class Bank

Bank bank = new Bank();

* Read from a file a list of customers, and add them to the Bank list
* You are going to have the following code in main:

Public static void main ( String[] args)

{

<put more code here>

//Sort the customers in ascending order based on the last and //first name

System.out.println(“sort the list”);

bank.menu(“6”);

bank.menu( “7”);

// print the customer list

System.out.println(“print the list”);

bank.menu( “7”);

//Withdraw $125 from Smith John account

System.out.println(“Withdraw $125 from Smith John account

”);

bank.menu( “3 : 125, Smith John”);

bank.menu( “7”);

//Withdraw $300 from Smith John account

System.out.println(“Withdraw $300 from Smith John account”);

Bank.menu( “3 : 300, Smith John”);

bank.menu( “7”);

//Deposit $500 to Clinton Hillery's account

System.out.println(“Deposit $500 to Clinton Hillery's account”);

bank.menu( “2 : 500, Clinton Hillery”);

System.out.println(bank);

//Remove Obama Barak from the list

System.out.println(“Remove Obama Barak from the list”);

bank.menu( “5 : Obama Barak”);

bank.menu( “7”);

//Deposit -$100 into George Bush account

System.out.println(“Deposit -$100 into George Bush account”);

bank.menu( “2 : -100 George Bush”);

//Withdraw -$50 from Biden Joe account

System.out.println(“Withdraw -$50 from Biden Joe account “);

bank.menu( “3 : -50 Biden Joe”);

bank.menu( “7”);

**//Display the customer list after each one of the above operations.**

* Continue displaying the menu until the user wish to stop - pressed 7 to quit (Do not ask the user after each operation if he want to continue yes/no, just display the menu)

Test your program with the following data:

Smith John (212)444-7866 325.60 saving

Biden Joe (716)111-3356 2000.53 checking

Obama Barak (310)777-1212 5000.32 checking

Boehner John (509)888-1234 6120.55 savings

Bush George (607)567-2345 200.00 savings

Clinton Hillery (212)456-2378 1000.50 checking

See the attached Coding Styles for your program.

**Due Date** :

March 4th **LAB (you have to demonstrate to me that the lab is working)**

**Hand In and Grading Criteria:**

For grading submit the following items in a **closeable** large envelop (no folders) with your name clearly stated outside:

Everything must be typed - ***no handwritten external documentation will be accepted.***

* A cover sheet with your name, project number, and section
* Your program design – UML, Javadocs, pseudocode, etc **(hard copies**)
* A **hard copy** of the source files
* A **hard Copy** of the **test cases** stapled together with your name marked on top
* A **hard copy** of the program output
* You will be required to show me that the programs are working

The following criteria will be used to grade your work.

* Program correctness – program runs successfully and as expected
* Adherence to specifications – appropriate control structures, modular design, etc
* Documentation – internal and external, meaningful variable names, etc
* Neatness / Organization – indentation, block formatting, etc

See the evaluation sheet attached to these specifications.

**Coding Style Example:**

Note the indentation, white space, and meaningful names.

/\*\* Program CSC 225 Prog1

\* Course Title: Introduction to Computer Science

\* Course number: CIS 225-xxx

\* Instructor: Sara Wexler

\* @author Your Name

\* @version 1.0, 1/10/200x

\*

\* Description: Program CIS 225 Prog1

\* This class computes the area of a circle. The radius is hard coded in the class.

\*

\* Input: radius

\*

\* Compute: area

\* radius times radius times PI (3.14159)

\*

\* @author Your Name

\* @version 1.0, 9/6/2003

\*/

public class ComputeArea

{

/\*\* Main method \*/

public static void main(String[] args) {

double radius;

double area;

// Assign a radius

radius = 20;

// Calculate the area

area = radius \* radius \* 3.14159;

// Display results

System.out.println("The area for the circle of radius " +

radius + " is " + area);

} // end for main method

} // end for class ComputeArea

/\*\*  
 \* Change the current capacity of this bag.  
 \* @param minimumCapacity  
 \* the new capacity for this bag

* Data is instance variable the array in this class  
   \* @postcondition  
   \* This bag's capacity has been changed to at least minimumCapacity.  
   \* If the capacity was already at or greater than minimumCapacity,  
   \* then the capacity is left unchanged.  
   \* @exception OutOfMemoryError  
   \* Indicates insufficient memory for: new int[minimumCapacity].  
   \*\*/  
   public void **ensureCapacity**(int minimumCapacity)  
   {  
   int biggerArray[ ];  
    
   if (data.length < minimumCapacity)  
   {  
   biggerArray = new int[minimumCapacity];  
   System.arraycopy(data, 0, biggerArray, 0,

manyItems);  
 data = biggerArray;  
 }  
 }