Project 02, Object-Oriented Programming, CSE 271, Spring 2020
Department of Computer Science and Software Engineering, Miami University

Class, File, Javadoc, JUnit

In this project, you will practice how to create classes and its methods, use file to store and retrieve data, write and generate Javadoc documentation, and test using JUnit library. Create a project in Eclipse named **Project_02.** You are going to design multiple classes in this project.

- **Class Address**:
  Create a class named **Address** which has the following private instance variables:
  - **int streetNumber:** This is an integer variable that holds the street number of the address.
  - **String streetName:** This field references a String object that holds street name.
  - **String city:** A String object that holds the name of the city.
  - **String state**: A String object that holds two letter abbreviations of the states.
  - **String zipCode**: A String object that holds five-digit ZIP code.

  The Address class has the following methods:
  - public Address()
    Initializes the instances variables to default values, strings to null and integer to zero.
  - public Address(int streetNumber, String streetName, String city, String state, String zipCode)
    Initializes the instances variables using the received parameters.
  - public Address(Address address)
    Initializes the instances variables using address object's instance variables.
  - Accessor and mutator methods for all instance variables:
    - public void setStreetNumber(int streetNumber)
    - public void setStreetName(String streetName)
    - public void setCity(String city)
    - public void setState(String state)
    - public void setZipCode(String zipCode)
    - public int getStreetNumber()
    - public String getStreetName()
    - public String getCity()
    - public String getState()
    - public String getZipCode()
  - public boolean equals(Address address)
    The method returns true if the two addresses have the same street number, street name, city, state, and ZIP code. False, otherwise.
  - public String toString()
    The method returns a String representing the address object that includes street number, street name, city, state and ZIP code. When this method is called, it returns a String like "2000 Main Street, Santa Monica, FL, 30309".

- **Class Customer**:
  Create a class named **Customer** which has the following private instance variables:
  - **int id**: This is an integer variable that holds the ID of the customer. The IDs are autogenerated and starts from 1000. The very first customer object will have the ID 1000. Each customer object after that will get an ID incremented by 1. For example, 2nd and 3rd customer objects are going to have ID 1001 and 1002 respectively.
  - **String name**: The name field references a String object that holds the name of the customer.
  - **Address address:** The address field is an Address object that holds the address of the customer.
  - **String ssn**: The ssn field references a String object that holds the social security number of the customer. It has the format "XXX-XX-XXXX". For example, "123-45-6789".

The class Customer has the following methods:

- public Customer()
  Assigns an ID to the **"id"** field (starts at 1000) and other instances variables **null.**
- public Customer(String name, Address address, String ssn)
  Assigns an ID to the **"id"** field (starts at 1000) and initializes the instances variables using the received parameters.
- public Customer(Customer customer)
  Initializes the instances variables using customer object's instance variables. The "id" field should be autogenerated not copied from the received object.
- Accessor and mutator methods for all instance variables except the mutator method of ID:
  - public void setName(String name)
  - public void setAddress(Address address)
  - public void setSSN(String ssn)
  - public int getID()
  - public String getName()
  - public Address getAddress()
  - public String getSSN()
- public boolean equals(Customer customer)
  The method returns true if the two customers have the same name, address, and SSN. False, otherwise.
- public String toString()
  The method returns a String representing the customer object that includes ID, name, address, and ssn. When this method is called, it returns a String like "1000, Sarah Williams, 2000 Main Street, Santa Monica, FL, 30309, 123-45-6789".

- **Class Account**:
  Create a class named **Account** which has the following private instance variables:
  - **int id**: An integer variable that holds the ID of the account. The IDs are autogenerated and starts from 1000. The very first account object will have the ID 1000. Each account object after that will get an ID incremented by 10. For example, 2nd and 3rd customer objects are going to have ID 1010 and 1020 respectively.
  - **Customer customer**: A customer object that holds the information of the account holder.
  - **double balance**: A double variable that holds the current balance of the account. Balance cannot be a negative number.

  The class Account has the following methods:
  - public Account(Customer customer)
    Assigns an ID to the "**id**" field and initializes the customer using the received parameter. It also initializes the balance to zero.
  - public Account(Customer customer, double balance)
    Assigns an ID to the "**id**" field and initializes the instance variables using the received parameters.
  - public Account(Account account)
    Initializes the instances variables using account object's instance variables. The "id" field should be autogenerated not copied from the received object.
  - Accessor and mutator methods for all instance variables except the mutator method of ID:
    - public void setCustomer(Customer customer)
    - public void setBalance(double balance)
    - public int getID()
    - public Customer getCustomer()

  - public double getBalance()
- public void deposit(double amount)
  Deposits money to the account, i.e. adds the received parameter "**amount**" to the balance.
- public boolean withdraw(double amount)
  Withdraws money from the account if it has enough balance, i.e. deducts the received parameter "**amount**" from the balance if **balance** is greater than or equals to **amount**. Returns true if successful, false otherwise.
- public boolean equals(Account account)
  The method returns true if the two accounts have the same ID and customer. False, otherwise.
- public String toString()
  The method returns a String representing the account object that includes ID, customer information, and balance. When this method is called, it returns a String like "1010, 1001, Sarah Williams, 2000 Main Street, Santa Monica, FL, 30309, 123-45-6789, 9099.03". The first number in the string is the account id, the second number is the customer id and the last number is the balance in the account.

- **Class Project02Driver**
  Create a class named Project02Driver which has the following methods:
  - public static Account[] readAccountsFromFile(String fileName)
    Reads all the accounts information from the file name "accounts.txt" and returns an array of Account objects. Returns null if there is no information in the file. We assume the file contains information of 10 accounts. The account information is stored as string in the text file. Each line has the information of a single account in the following format:
    "1010, 1001, Sarah Williams, 2000 Main Street, Santa Monica, FL, 30309, 123-45-6789, 9099.03". Remember that the account and customer ids are autogenerated. You should discard first two ids when you create new Account objects.
    Hint: You can use the split method from the String class to separate the single account information using comma as a delimiter. Split the line into an array of strings. Use the strings to create an account object.
  - public static boolean writeAccountsToFile(Account [] accounts, String fileName)
    Writes the information from the accounts array to the file as text. Returns false if it fails to write to the file, true, otherwise.
  - public static void main(String [] args)
    Call and test readAccountsFromFile() and writeAccountsToFile() methods.
- JUnit Tester Classes
  - Create a class named AddressTester to test Address class.
  - Create a class named CustomerTester to test Customer class.
  - Create a class named AccountTester to test Account class.

  In your Tester classes you have to test all the methods of the classes including constructors, accessors and mutators. You can create multiple objects of the class for testing. When you test you need to make sure that you think about all possible scenarios for each of the methods. Review lab assignment 5 and its solution (posted on Canvas) to see examples of JUnit test cases. Also, review posted solutions to other labs to complete this project.

Javadoc Style Comments:
You have to make Javadoc style comments for all classes and methods including parameter and return description. Once you write all the Javadoc style comments generate Javadoc using Eclipse's "Generate Javadoc" option from the "project" menu. Include the Javadoc folder "doc" with your code submission.

Project 02, Object-Oriented Programming, CSE 271, Spring 2020
Department of Computer Science and Software Engineering, Miami University

Class, File, Javadoc, JUnit

Important Note:
Make sure the file names are correct and code is well commented. No late submission. You will get zero for late submission.

Submission:
Submit java files and Javadoc folder "doc" to the appropriate submission folder on the Canvas by the due time. You can zip all the java files and "doc" folder together and submit one zip file.

Grading Rubric:

Address	
Constructor (each worth 1 point)	3
Accessors and Mutators (each worth 0.5 points)	5
equals() method	3
toString() method	3
Customer	
Constructors (each worth 1 point)	3
Accessors and Mutators (each worth 0.5 points)	3.5
equals() method	3
toString() method	3
Account	
Constructors (each worth 1 points)	3
Accessors and Mutators (each worth 0.5 points)	2.5
equals() method	3
toString() method	3
deposit() method	3
withdraw() method	3
Project02Driver	
readAccountsFromFile() method	9
writeAccountsToFile() method	9
main() method Test readAccountsFromFile() and writeAccountsToFile() methods (3 points each)	6
Testers	
Test constructors (each worth 1 points)	9
Test accessors and mutators (each worth 0.5 points)	11
Test equals and toString methods (each worth 1 points)	6
Test deposit and withdraw methods (each worth 3 points)	6
Javadoc	
Javadoc style comments (for each method 0.25 points and for each class 0.25) [Only deduction]	0 [-5]
Submitted correct Javadoc files [Only deduction]	0 [-5]
Total	**100**