**LAB #1 - Creating Java Applications with Eclipse IDE**

**Due Date: Week 3 (demo of your code in class) – 23 January, 2017**

**Value: 5%**

**Maximum points: 20 points**

**Purpose:** The purpose of this Lab assignment is to:

1. Become familiar with Eclipse IDE
2. Create simple Java Applications
3. Create and use a simple Java classes

**References:** Read the course’s text “Java How to program, 10th edition”, chapters 1 to 3 and the lecture notes. This material provides the necessary information that you need to complete the exercises.

**Instructions**: Be sure to read the following general instructions carefully:

This lab should be completed individually by all the students. You will have to demonstrate your solution in a scheduled lab session and submit the project **through dropbox link on eCentennial**.

You must name your Eclipse project according to the following rule:

**YourFullName\_COMP228\_sectionNumber\_LabNumber**

Example: **JohSmith\_COMP228\_002\_Lab1**

Each exercise should be placed in a separate package named *exercise1*, *exercise2*, etc.

Submit your assignment in a **zip file** that is named according to the following rule:

**YourLastName\_COMP228\_sectionNumber\_LabNumber.zip**

Example: **JohSmith\_COMP228\_002\_Lab1.zip**

Apply the naming conventions for variables, methods, classes, and packages:

- *variable names* start with a *lowercase* character

- *classes* start with an *uppercase* character

- **packages** use only *lowercase* characters

- *methods* start with a *lowercase* character

**Exercise #1: (10 points)**

Create a shell(blueprint) **– 2 points**

1. Create a public class named *Patient*.
2. Create some **private** instance variables to hold the following information (decide on data type yourself):
3. *patient id*,
4. *first name,*
5. *last name,*
6. *age*
7. Provide public *getter* and *setter* methods
8. Provide a *getPatientInformation* method to return patient’s information in a nicely formatted string. Use toString method.
9. Provide a *default* *constructor*
10. Provide a *multi argument* *constructor* that allows to initialize all of the patient data.

Internal documentation: **– 1 point**

1. Use *traditional comments* inside your patient class and driver class to provide explanation of what method and other statements do (at least 4 comments)
2. Use *Javadoc comment* at the beginning of each class with your name as author and short description of the class

Write a driver class for the patient **– 3 points**

1. Create a driver class PatientDriver with main method
2. Instantiate one patient and set the values with the help of the multi argument constructor.

The values must be the following:

1. patient id is *1*
2. first name is *Lucy*
3. last name is *Brown*
4. age is 25
5. Create another patient object with a default constructor. No values are set.
6. Now prompt user for data for **each** field of your second object patient. “Please enter patient ID: “, “Please, enter first name: “, etc… Once user enters the data call setter method and set the instance variable. If value of age provided by the user is less than 19, then prompt user again to enter the age: “Age must be 19 or older. Please enter age again: “ . Limitation – the second time user must enter the correct value for age.
7. You will have two patients in total. When user completes entering all of the data, prompt user to enter first name of the patient. “Please enter first name of the patient that you would like to see the details about: “. If user enters Lucy then display the information for that patient. If user enters some other name then display the patient only if it is one of the patients that you created otherwise display message “Sorry! No patient found!”

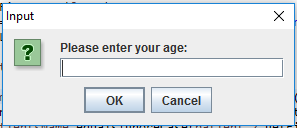
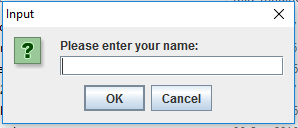
Use the *JOptionPane* for input/output in the driver class. **– 2 point**

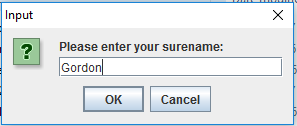
Write a unit test to test your patient getters and setters – **1 points**

1. Test negative scenario
2. Test positive scenario

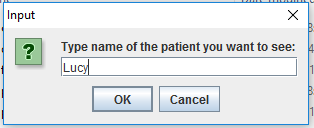
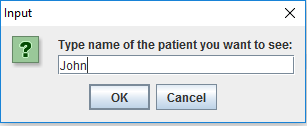
Make a UML diagram of your patient class – **1 points**

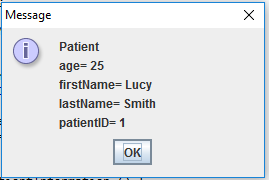
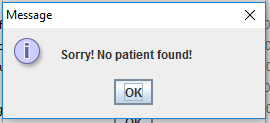
**Examples for prompts:**



**Example for no patient found: Example for Lucy:**

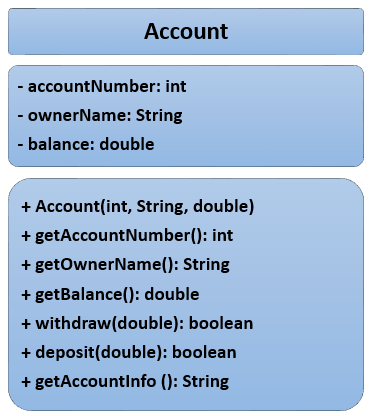




**Exercise #2: (10 points)**

In this exercise you will develop a Java application that will have one object.

Create a shell(blueprint) of Account class based on the UML diagram below **– 2 points**



Internal documentation: **– 1 points**

1. Use *end-of-line comments* inside your account class and driver class to provide explanation of what do method and other statements do (at least 4 comments)
2. Use *Javadoc comment* at the beginning of each class with your name as author and short description of the class

Write a driver class for the bank account. **– 3 points**

1. The driver class (main class) instantiates the account.
2. Create an object of account type and set the values with the help of the constructor as follows:
3. Account number is *1001*
4. Owner’s name is *Bill*
5. Balance is *100.00*
6. The driver class then displays the information by calling *getAccountInfo* method
7. The driver class prompts the user to select an option: 1 or 2

option 1 – withdraw from account

option 2 – deposit to the account

1. Depending on the option user selects display a prompt to user “How much would you like to withdraw:” or “How much would you like to deposit:” And call corresponding method: *withdraw* or *deposit*

Add **a validation rule** - if a user enters value for withdrawal greater than the value of the balance then do not modify balance and display message to the user “Withdrawal amount exceeded account balance.”

**Another validation rule** for deposit: if user enters negative value or 0 display message “Incorrect amount. Please try again.”

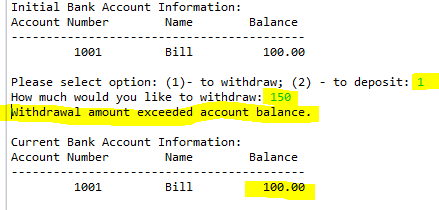
1. After user does the transaction display all the info about the account to the user again. There should be change in the balance value.

Use **console** to provide input/output in the driver class. **– 2 points**

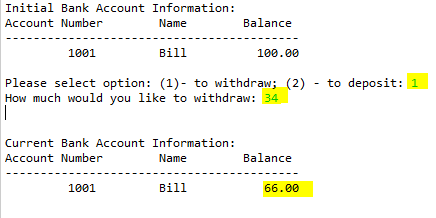
Write a unit test to test the following: – **1 point**

1. Test withdraw method - negative scenario
2. Test withdraw method - positive scenario
3. Test deposit method - negative scenario
4. Test deposit method - positive scenario

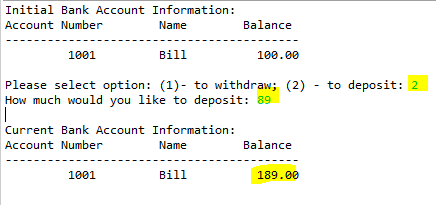
**Example for the validation:**



**Example for Withdraw:**



**Example for Deposit:**



**Evaluation:**

|  |  |
| --- | --- |
| **Functionality** | **%** |
| Correct implementation of classes (instance variable declarations, constructors, getter and setter methods, etc.) | 35% |
| Correct implementation of driver classes (declaring and creating objects, calling their methods, interacting with user, displaying results) | 45% |
| Comments, correct naming of variables, methods, classes, etc. | 10% |
| GUI | 10% |
| **Total** | **100%** |

Late submissions:

**20% deducted** for each day late.