

# Programming Assignment 7/8

**Scenario:**

After graduating from GMU, you are running your own consulting firm, providing computer services around the metro area. Your company charges clients by the hour, and your employees sometimes need to visit client sites. To do so, they have the option of driving their own car or taking a taxi or other ride sharing service.

Your primary client wants a weekly summary of each consultant’s billable hours, and any travel expenses incurred each week (such as Personal Car Mileage or Ride Services).

Each billable item should track the employee to whom it relates, and the day of the week on which the charge was incurred, and be rounded to the nearest dollar.

Billable Hours should include the number of hours worked by the employee, and their hourly rate.

Personal Car travel should be tracked in terms of distance, and billed at the rate of $1.00 per mile. Personal Car travel should also keep a list of the names of any other employees that rode in the same vehicle, and no one may bill for travel (Personal Car or Ride Service) on the same day that they are a passenger in another car. Assume that no car can hold more than 4 people, including the driver.

Ride service billing should include the price and the name of the service (Lyft, Uber, Yellow Cab, etc).

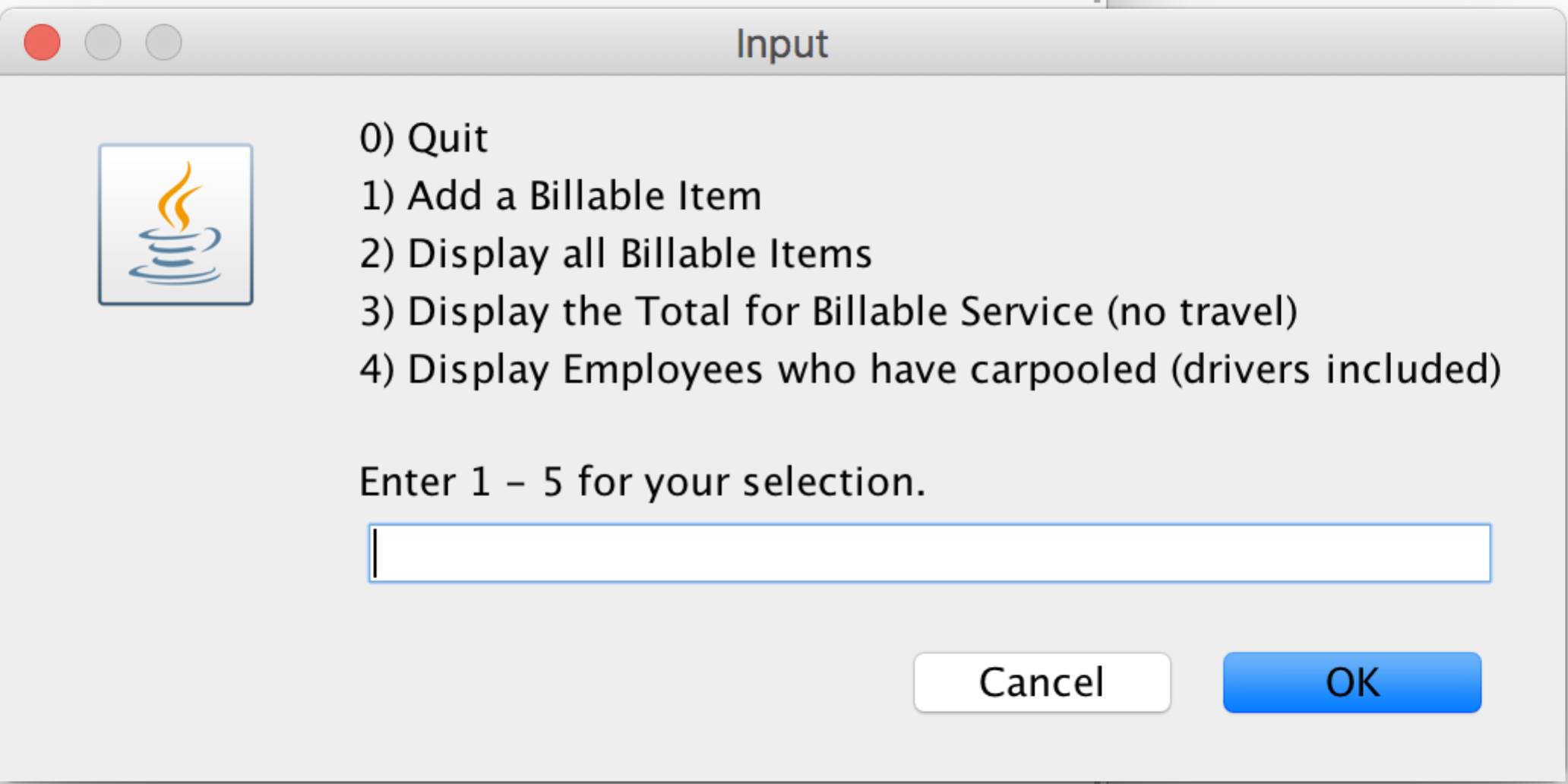
For your own benefit, you want to be able to display the total due for Billable Hours (not travel), and to display the names of employees who have ridden with others (drivers included) to encourage carpooling.

Do not assume that employees travel every day, or work for this client every day.

Create an efficient, object-oriented solution, using good design principles, with a polished user interface that will allow you to generate a weekly summary for your client. At present, you have five employees, but anticipate steady growth. You should plan the system’s capacities accordingly (e.g. each employee could feasibly user Uber twice a day, five or six days a week, plus bill time, etc). Naturally, you should also prohibit the user from exceeding the system’s capacity.

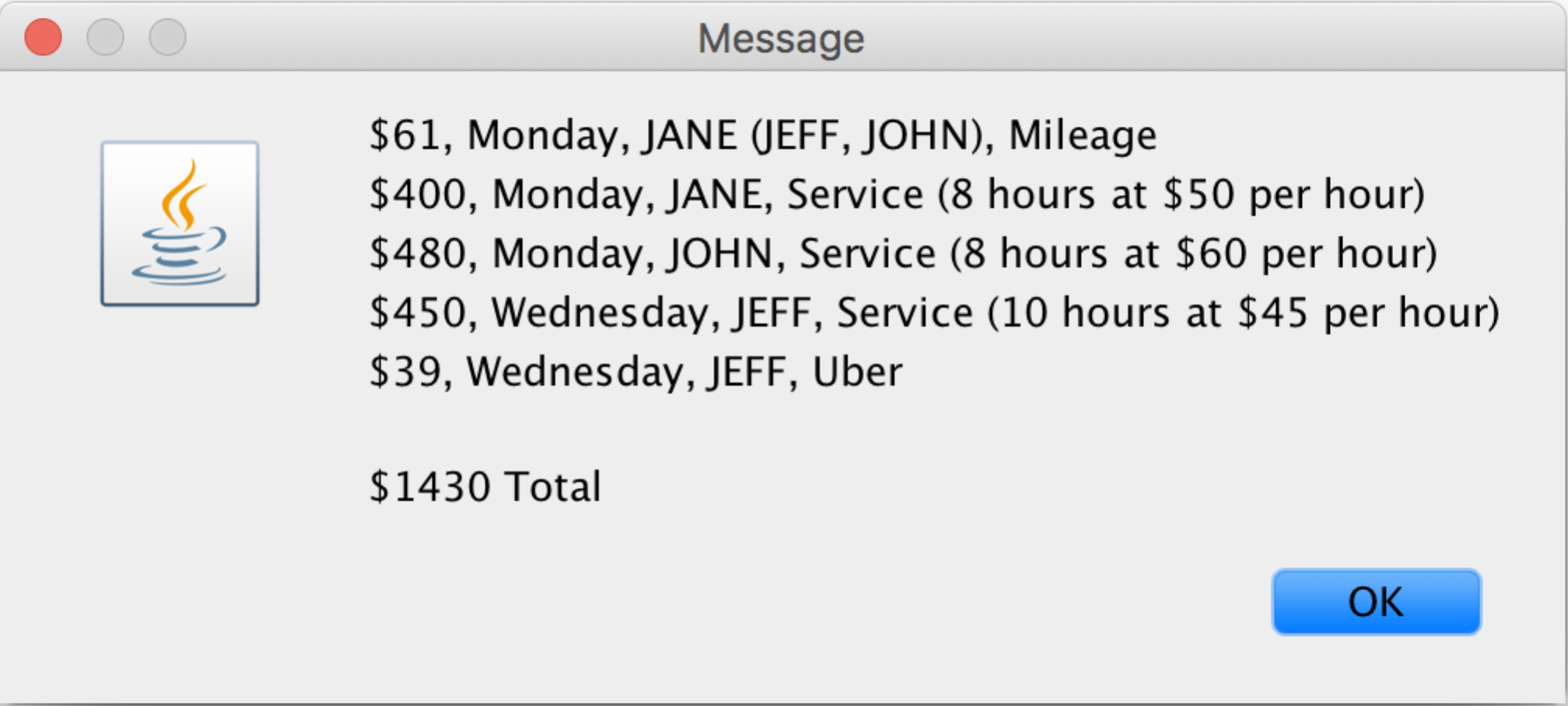
**Program Requirements**:

The user should initially be presented with a menu like the following, to prompt for which program function they would like to use (by number):



The “Add a Billable Item” option should allow a user to select the type of type of billable item (Service Hours, Personal Car Mileage, or Ride Service. After selecting the type, the user should be prompted to enter all the appropriate related information as described above.

The “Display all Billable Items” option should provide a detailed display of all billable items and information and a total, like the following (note the additional passengers with Jane on the first line).



The “Display the Total for Billable Service” option should display a dollar figure equal to the total of billing charges for all employees (i.e. total of each employee’s hours multiplied by their billing rate). *Note that this amount does not include travel*.

The “Display Employees who have carpooled” option should display the name of each employee and driver who has carpooled (once per name in the case where someone has carpooled multiple times).

The “Quit” option will allow the program to terminate. When any other option is selected, the program should execute appropriately, and then return to the main menu.

Whenever an invalid choice is made, the user should be warned and returned to the same prompt.

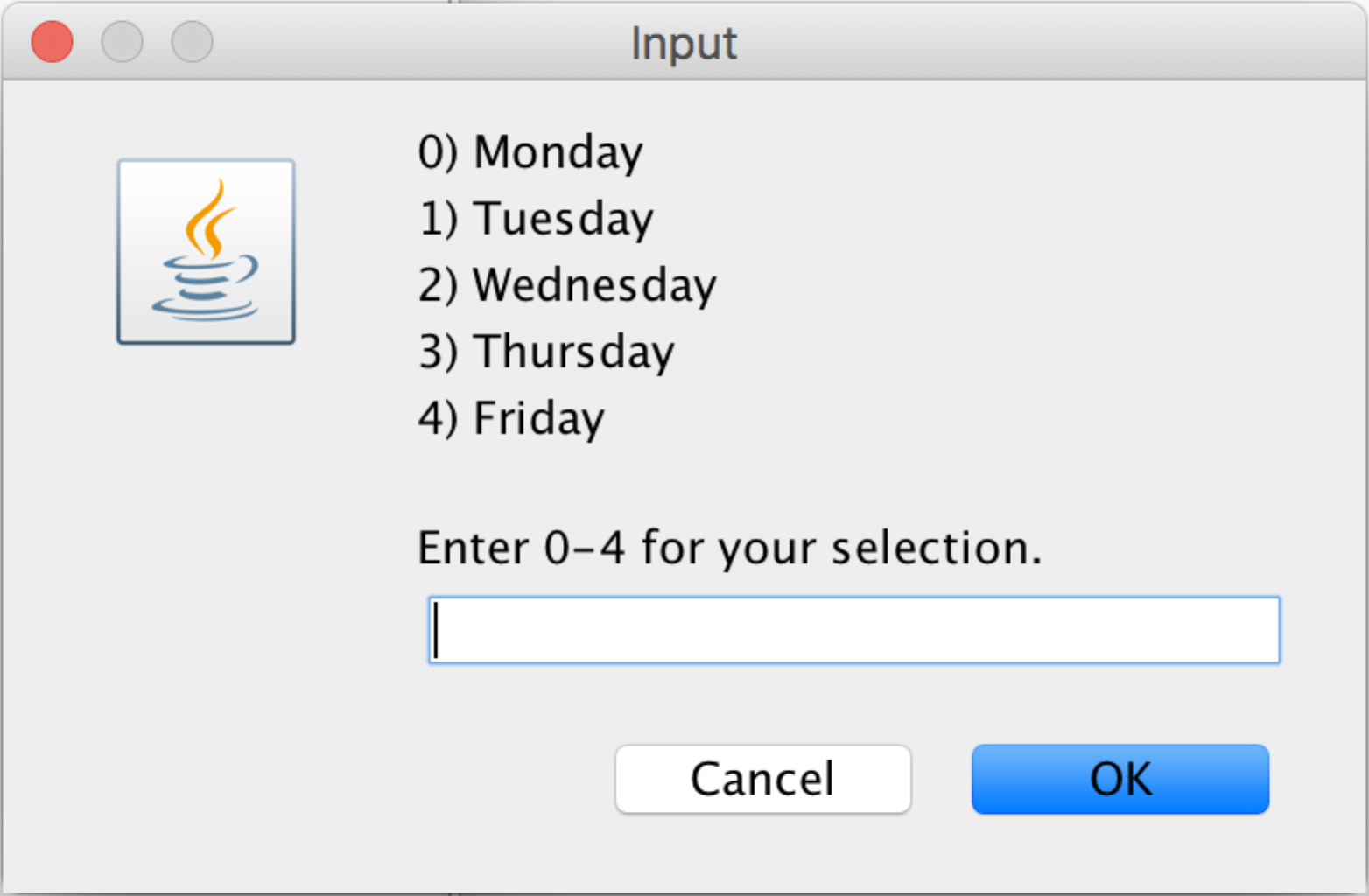
All billable items—regardless of type—should be stored in a single array.

Mutators and constructors should disallow invalid inputs (e.g. hours, rates, mileage, etc, should be greater than zero, and String inputs should not be blank or only spaces). Additionally, all String values should be stored and displayed in lower case, except for the first letter which should be upper case (e.g. Like this), regardless of how it was entered by the user.

As a matter of good practice, your class definitions should include equals() and toString(), whether or not your working solution uses them.

**Optional**:

To facilitate usage, testing, etc, you may prompt the user with specific inputs, as illustrated below (with the caveats that follow):



If you choose to do so, then your inputs, you must use an array to hold the options. For example:

String[] myInputChoicesAre = {“Monday”,”Tuesday”,”Wednesday”,”Thursday”,”FRIDAY”};

This is to facilitate changes, such as adding Sunday and Saturday in the example above, and to ensure that nothing else is “hard coded” in your system (whomever is grading your program may wish to test with values that you have not included). Additionally, do not rely on the array values to be valid or correct (e.g. in the example above, your objects should never display “FRIDAY,” only “Friday”). In other words, do not rely on data entry to validate your constructor and mutator inputs (remember, part of object-oriented programming is reusability, and in theory your entity classes could be re-used by other programs that might not provide valid data.

From a practical standpoint, this approach will save you from re-typing the days of the week, employee names, etc over and over as you develop and test your program!

**Other Requirements:**

* You must devise an object-oriented solution, including constructors, accessors (including a toString() method), mutators, and special purpose methods as appropriate. No points earned for a procedural solution.
* Constants should be defined and used when appropriate as well.
* Your solution must not import any Java library other than JOptionPane.
* Your solution must demonstrate use of arrays.
* Your solution must demonstrate the concepts of abstraction, inheritance, and polymorphism.
* As a matter of good practice, your class definitions should include equals() and toString() methods, whether or not your working solution requires them.
* Your controller class should use good design principles, and include purpose-specific methods other than main().
* You must use JOptionPane for all input and output.
* Do not use System.exit, or any variant to terminate the program. The program’s logical structure should allow the thread-of-control to reach the last program statement and terminate.
* Your solution may not use any language features not covered during IT 106 or this semester’s IT 206 without prior authorization from your instructor. **Doing so may lead to a substantial grade penalty, a grade of zero, or an Honor Code inquiry if we suspect that the work demonstrated beyond your expected level of ability.** When in doubt, ask!

**Programming Assignment 7: Solution Design**

1. List and describe the purpose of each class that will be needed to solve this problem
   1. You must separately identify (list) and describe the purpose of each class. One or two sentences per class should be sufficient.
2. *Data Definition Class(es)* - Create a detailed UML Class Diagram, listing and explaining all class variables, accessors, mutators, special purpose methods, and constructors associated to each data definition class.
3. *Implementation Class* - Create a table that lists all methods that will be used to create the implementation class. For each method identified, provide the following:
   1. A 1-2 sentence describing the purpose of the method
   2. A list of the names, data types, and brief description (1-2 sentences) for each input variable into the method, if there are any
   3. The name and data type of the variable to be returned from the method, or void if nothing will be returned

**Note:** You do not need to provide any pseudocode. However, you should create a plan for yourself as to how you will perform the logic for each method. If you do not do this, you will have great difficulty in completing the solution implementation.

An example format to use for this table is as follows. You can format this table in any way you’d like, so long as the information is clearly presented. Completing this table correctly will help you build your documentation and code for your solution implementation.

**Method:** getMagazine

**Purpose:** The method will allow for the creation and population of a Magazine object based on user input of a title, cost, and number in stock

**Inputs:** none

**Return:** magazine : Magazine – The Magazine object created and populated from user input

**Method:** checkout

**Purpose:** The method will allow for the purchase of a magazine. It takes into account the sales tax that will be charged as part of the cost calculation

**Inputs:** magazine : Magazine – The magazine to be purchased

salesTax : double – The sales tax percentage to be charged as part of the checkout

**Return:** void

Upload a Word document containing only items above to Blackboard.

|  |  |
| --- | --- |
| **Grading Criteria** | |
| **Requirement** | **Points** |
| List and describe the class(es) needed to solve the problem | 10 |
| Data Definition Class Design – Detailed UML Diagram | 40 |
| Implementation Class Design – Table Listing All Methods | 50 |

**Programming Assignment 8: Solution Implementation**

Write a well-documented, efficient Java program that implements the solution design you identified. Include appropriate documentation as identified in the documentation expectations document.

**Note:** You must use the JOptionPane class for input/output. Additionally, if you use System.exit as shown in the textbook, it may only be used as the absolute last line in the program. You may not use System.exit, or any variant that exits the program in the middle of the program. The program should be designed to only exit once the algorithm has finished.

To Blackboard, **submit ONLY ONE .zip file** containing all of the .java files part of your submission for your solution implementation. As you are naming your Java files, make sure they are indicative to the purpose of the file. **Do not include your name in the file name or use generic names, such as Assignment4.java**. Your .zip file should contain only the .java files in your solution. Be careful that you do not submit .class files instead of .java files.

**Warning!** You must submit **ONLY** **ONE** .zip file containing **ONLY** your .java files. Failure to follow this instruction precisely will result in a 10 point deduction of the assignment score. **No exceptions!  
*Why is this important?* The goal is to teach you how to properly package your IT solutions into a “customer-friendly” format while paying attention to “customer” requirements provided to you.**

**Your program must compile using jGrasp**. Any final program that does not compile, for any reason, will receive an automatic zero. Other IDEs often place in additional code that you are unaware of, doing too much of the work for you. **You are strongly discouraged from using IDEs other than jGrasp.**

|  |  |
| --- | --- |
| **Grading Criteria** | |
| **Requirement** | **Points** |
| Implementation of object-oriented Java program, using efficient practices, such as the use of constants, good variable names, information hiding, no redundant code, etc. | 70 |
| Appropriate objective-style documentation | 10 |
| Appropriate intermediate comments | 20 |