**** Modifications to this Project Sheet may be needed and will be announced in class ****

CIS 200 (Scholars): Project 5 (60 + 3 points)

Due MONDAY, Oct 16th by 11:59pm

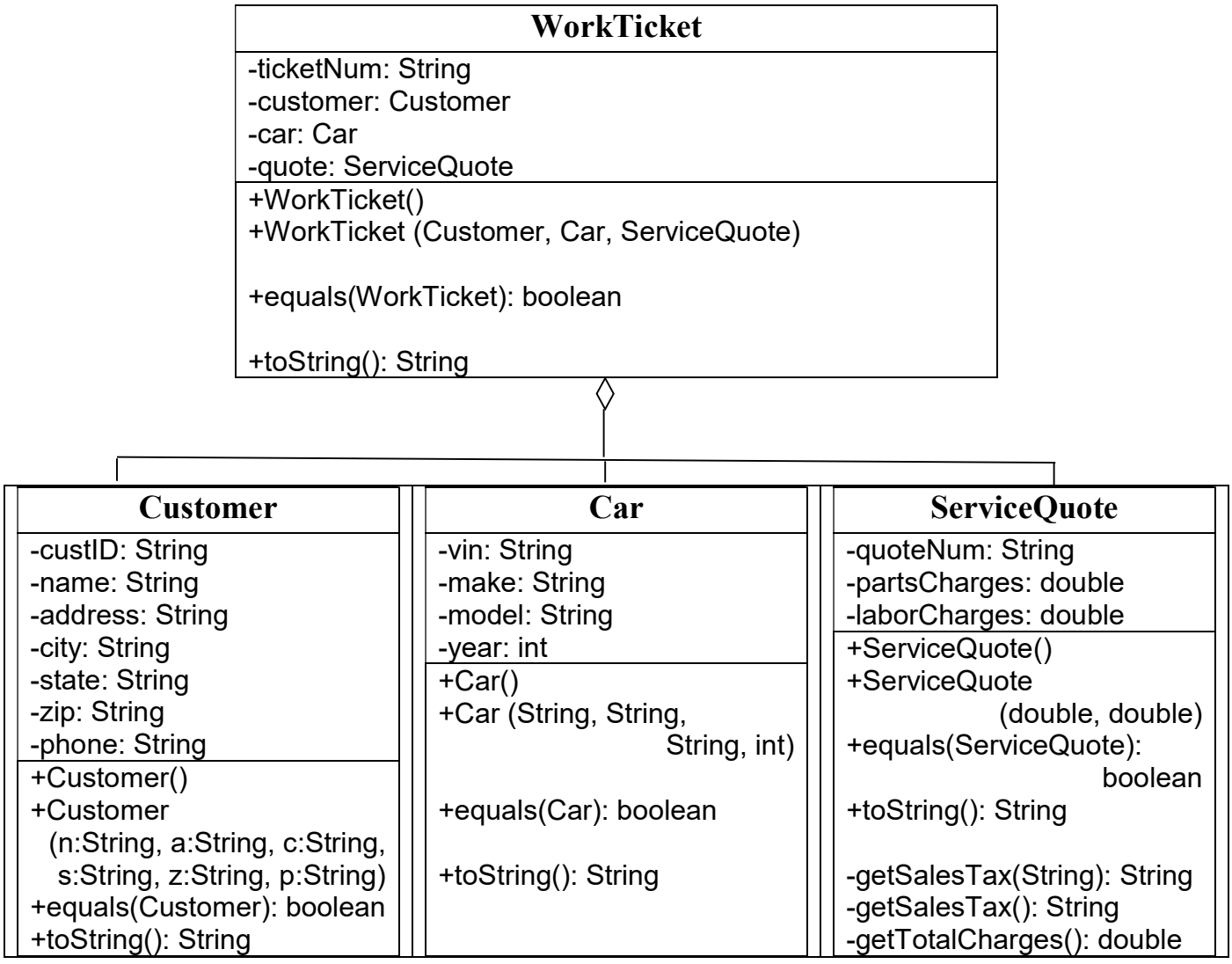
Reminder: Programs submitted after the due date/time will be penalized 10% for each day the project is late (not accepted after 3 days, i.e. Midnight, Thu, Oct 19th)

Reminder: As stated in the *Academic Integrity Policy* in the syllabus for this course, unless otherwise stated, **ALL projects are intended to be done *individually*. Do NOT share your code with anyone.** If it is not your work, don't submit it as yours. Refer to the policy within the course syllabus which states, ***"If two (or more) students are involved in ANY violation of this policy,* at a minimum, *ALL* students involved receive a *zero*** for the assignment and the offense is officially reported to the *KSU Honor Council*. The second offense results in a failing grade for the course and possible suspension from the university (this decision is made by the *K-State Honor Council*)."

Note: If you use code that YOU did not develop (i.e. from another text or found it on the internet) **you must cite your source** in a comment above the line(s) of code that is not your own work, otherwise it will be considered **plagiarism** (*representing others' work, whether copyrighted or not, as one's own*).

Assignment Description:

As discussed in class, applications usually implement *multiple* classes and include ***aggregation***, which "*occurs when an instance of a class is a field in another class*." (Gaddis, p.517). For this project, implement (i.e write the code) the classes described in the UML Class Diagrams shown below (similar to Section 6.9, pp.384—390, of your textbook.) ***get/set* methods may be added as needed**.



- For **custID**, use the first 4 letter of the user's name followed by a 4-digit randomly generated number. You can assume the user will have at least 4 letters in their last name. (Ex: smit4857)
- For **quoteNum**, use two randomly generated upper-case letters followed by a 4-digit randomly generated number. (Ex: AQ9958)
- For **ticketNum**, use the first 2 letters and first two digits of the custID and the 4-digits of the quoteNum. (Ex: sm489958)
- Default *no-arg* constructor for each class can be whatever you want (even a 'dummy' constructor), as long as it is included.
- The **getSalesTax** method is **overloaded**. If passed a String, it will simply pass back $0 plus that String to be used in displaying the sales tax. EX: "$0 (Labor only – no sales tax). If passed nothing, formats and returns *as a String* the sales tax of the combined *parts* and *labor* charges (EX: $76.32). Sales Tax rate will be **6.5%** (use a constant)

equals() methods

- In the **Customer** class, two objects are equal if they have the same *custID*.
- In the **Car** class, two objects are equal if they have the same *Vehicle Identification Number* (vin).
- In the **ServiceQuote** class, two objects are equal if they have the same *quoteNum*.
- In the **WorkTicket** class, two objects are equal if they have the same *ticketNum*, *Customer*, *Car*, OR *ServiceQuote*. (Hint: Call the *equals* method of each of these classes within this method)

toString() methods

- In the **Customer** class, simply display each data property on an individual line. Preceded ID with an identifying label. (Ex: Customer ID: smit4857).
- In the **Car** class, display the Vin #, preceded by an identifying label. (Ex: VIN #: A92847fGt1263) followed by the *year make model* (on a separate line) (Ex: 2006 Ford F150)
- In the **ServiceQuote** class, display each data property on an individual line plus the Sales Tax and the Total Charges, preceded by an identifying label. Display all currency values in the format $#,###.##. (Hint: One way to do this is *DecimalFormat*)
- In the **WorkTicket** class, display the ticket number followed by the *Customer* info (call toString method), followed by a blank line, the *Car* info (call toString), a blank line, and the *ServiceQuote* info (call toString).

Your output will look like the following (2 Records shown below):

Ticket Number: sm489958 Customer ID: smit4857 Smith, Harold 1313 Mockingbird Lane, Manhattan, KS 66502 785-532-7768 VIN #: A92847fGt1263 2006 Ford F150 Quote Number: AQ9958 Parts: $525.67 Labor: $648.45 Sales Tax: $76.32 Total Estimated Charges: $1,250.44	Ticket Number: jo297338 Customer ID: jone2938 Jones, Sally 1710 Platt St. Manhattan, KS 66506 785-532-2211 VIN #: Q93847kAm3827 2013 Honda Accord Quote Number: ZT7338 Parts: $0 Labor: $234.78 Sales Tax: $0 (Labor only – no sales tax) Total Estimated Charges: $234.78

Once the initial 4 classes are coded and compile, create the following application class (**Proj5App**):

- Create and properly use an **Array** (or ArrayList) containing **WorkTicket** objects. You will allow the user to enter the information for as many *WorkTickets* as they would like (up to 50 – use a constant). You can choose how to end input. Store each object in the *Array*. Input will include all information needed to create a *WorkTicket* object.

Maximum of 1/2 Credit given if program does not utilize an **Array (or ArrayList) **of WorkTicket Objects** (which contain Customer, Car, and ServiceQuote objects) **

- Do not charge sales tax if parts are not ordered (i.e. call your overloaded salesTax method method)
- Before storing a **WorkTicket** object into the array, check for an existing *WorkTicket* object currently in your array (use your *equals* method). If a duplicate is found, simply display "*Duplicate Work Ticket found*" and have the user re-enter the information.
- **Once all input is entered,** use the *toString* method in **WorkTicket** class to display each object in the array. Display one object at a time, followed by a "*Hit enter to display the next WorkTicket*" message. When the user presses *enter*, display the next work ticket. When all have been displayed, display the message "*All work tickets displayed*."
- Lastly, **once all output is displayed,** allow the user the ability to **search** for as many work tickets as they desire (one at a time) by requesting a *Work Order Ticket Number*. **If found**, display as shown on the bottom of the previous page. **If not found**, display an appropriate message. It will be left up to you how you want to implement and end the search option.
- You don't have to worry about data validation with this project, but make sure and test with *no parts charges*, *no labor charges*, *no labor or parts charges*, with/without Sales Tax message.

ROUGH pseudocode might look like the following *(you do NOT have to structure yours like this!)*

Input: Loop until user no longer wants to enter in any more info (or maximum is reached)
-Read in info for a customer and create a *Customer* object
-Read in info for a car and create a *Car* object
-Read in info for a service quote and create a *ServiceQuote* object
-Create a temporary WorkTicket object
-Check for duplicate
-If not a duplicate, store WorkTicket Object in Array else display message and re-enter

Output: Loop through and display all objects stored in the Array

Search: Loop as long as the user wants to search and display found objects or '*not found'*

Documentation:
You must put a description of the project at the top of the file **and at the top of each method**.

Please use this template for the top of the file:

```
/**
 * (description of the project)
 * @author (your name)
 * @version (which number project this is)
*/
```

Please use this template for the top of each method:

```
/**
 * (description of the method)
 * @param (describe first parameter)
 * @param (describe second parameter)
 * (list all parameters, one per line)
 * @return (describe what is being returned)
 */
```

Requirements

Important: **ALL** files must compile **at the command-line** with the statement: **javac filename.java.** Project must run at the command-line with the statement: **java Proj5App.**

Make sure and test this before submitting. Submit ALL needed files and to submit the **CORRECT** files (i.e. the ones that compile). It is very important that you learn to submit what is needed to your *client*.

OPTIONAL Extra Credit Challenge: (5% extra credit - 3 points)

Store the four classes given on p.1 of this assignment sheet in a **package** called **joesGarage**. Then import all of these files into your application class. **Create an executable jar** (Prog5.jar) that can be used to test your program. Include this .jar file with your 5 class files inside your Proj5.zip folder.

Submission: Read these instructions carefully or you may lose points / get a ZERO on your project

Programs that do not compile will receive a grade of ZERO, regardless of the simplicity or complexity of the error, so make sure you submit the *correct* file that *properly compiles*.

To submit your project, first create a folder called ***Proj5*** and copy or move **ALL** created *.java* files into that folder. Then, right-click on that folder and select "***Send To → Compressed (zipped) folder***". This will create the file ***Proj5.zip***

Log-in to Canvas and upload your ***Proj5.zip*** file. **Only a .zip file will be accepted for this assignment in Canvas. Put your full name and Project 5 in the comments box.**

Important: It is the ***student's responsibility*** to verify that the ***correct*** file is ***properly*** submitted. If you don't properly submit the *correct* file, *it will not be accepted after the 3-day late period*. **No exceptions**.

Grading: Programs that do not compile will receive a grade of 0. If all needed files are not submitted, the program will NOT compile for the TA!

Programs that *do* compile will be graded according to the following rubric:

Requirement	Points
**Maximum of 1/2 Credit (30 pts.) if program does not utilize an Array (or ArrayList) of WorkTicket Objects (which contain Customer, Car, and ServiceQuote objects) **	
EXECUTION – Allows user to enter info for up to 50 WorkTickets the PROPERLY DISPLAYS (as shown on p.2) all objects stored in the array, one at a time, allowing user to press enter after viewing each object. Then allows user to SEARCH for as many work tickets as desired – properly displays found ticket or message "ticket not found".	**20**
CUSTOMER class (Code) – Correctly created from UML with correct PRIVATE data properties, constructors, and correctly defined *equals* and *toString* methods	**6**
CAR class (Code) – Correctly created from UML with correct PRIVATE data properties, constructors, and correctly defined *equals* and *toString* methods.	**6**
SERVICEQUOTE class (Code) – Correctly created from UML with correct PRIVATE data properties, sales tax constant, constructors, and correctly defined *equals* and *toString* methods plus PRIVATE *getTotalCharges* and PRIVATE OVERLOADED *getSalesTax* methods.	**10**
WORKTICKET class (Code) – Correctly created from UML with correct PRIVATE data properties, constructors, and correctly defined *equals* and *toString* methods that utilizes the *equals* and *toString* methods of Customer, Car, and ServiceQuote classes.	**8**
PROJ5APP class (Code) – Correctly creates and stores Work Ticket objects in array, array size constant, properly uses *equals* method to avoid storing duplicates and *toString* method to display output. Includes a loop to allow the user to search for Work Orders.	**7**
Proper Documentation on both the File Header of EACH CLASS and ON EACH METHOD (Purpose of each class should differ) / Proper Submission with full name & project # in Comment box. Correct Filenames	**3**
EXTRA CREDIT: Store 4 classes in a package called ***joesGarage***. Import these files into the application class. Include an executable .jar file that can be used to run/test your program.	**+3**
Minus Late Penalty (10% per day)	
Total	**60+3**