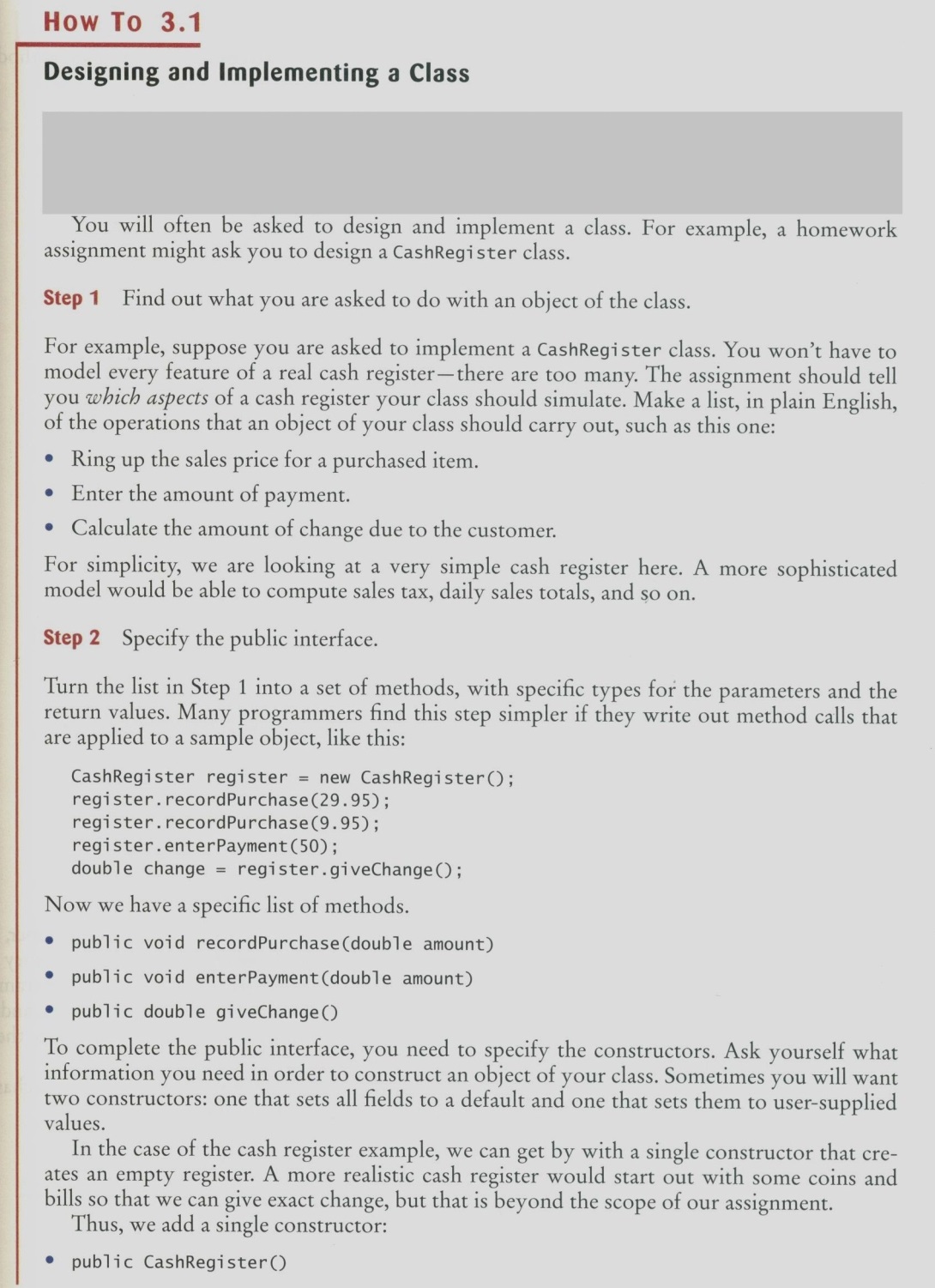
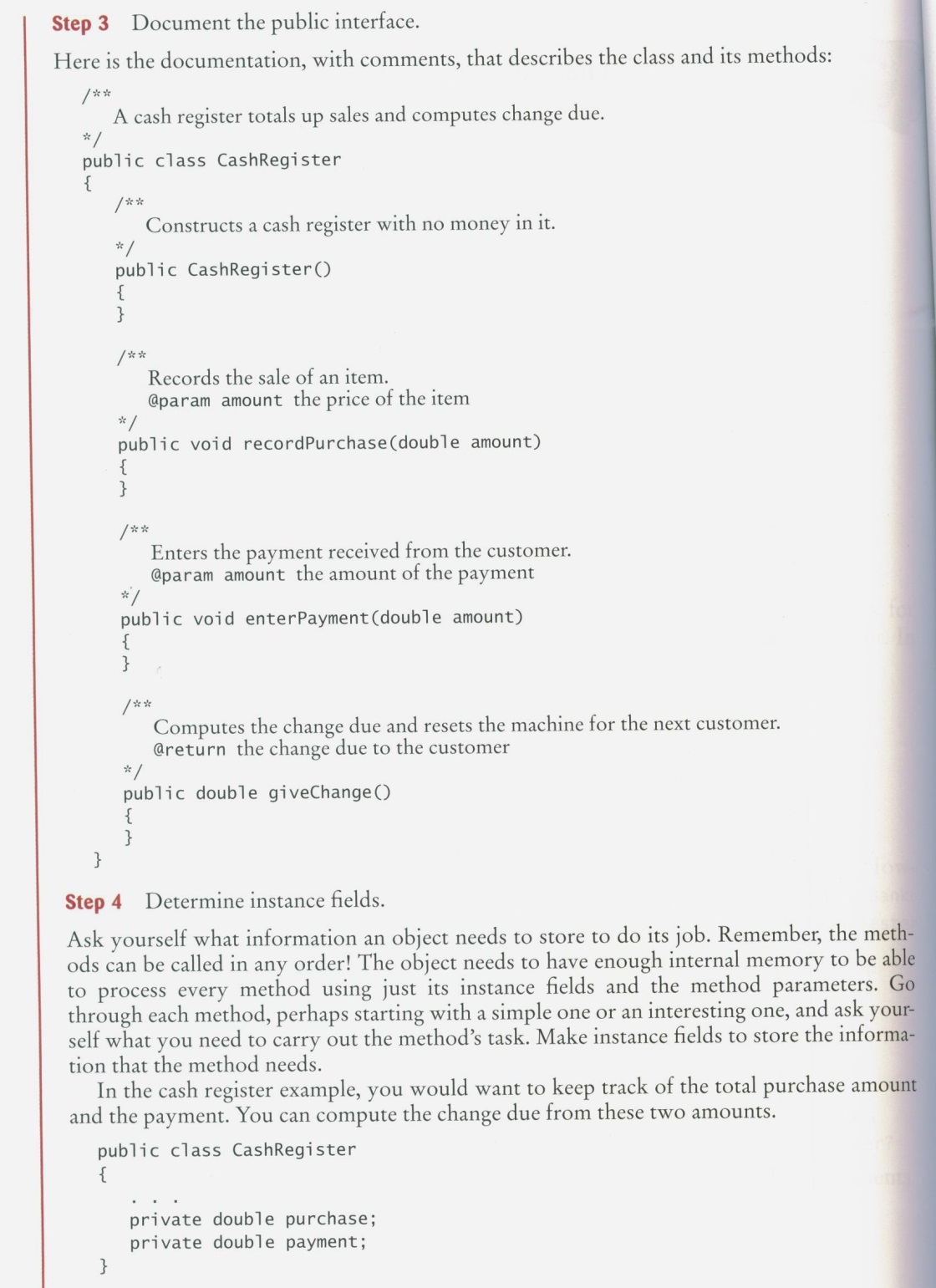
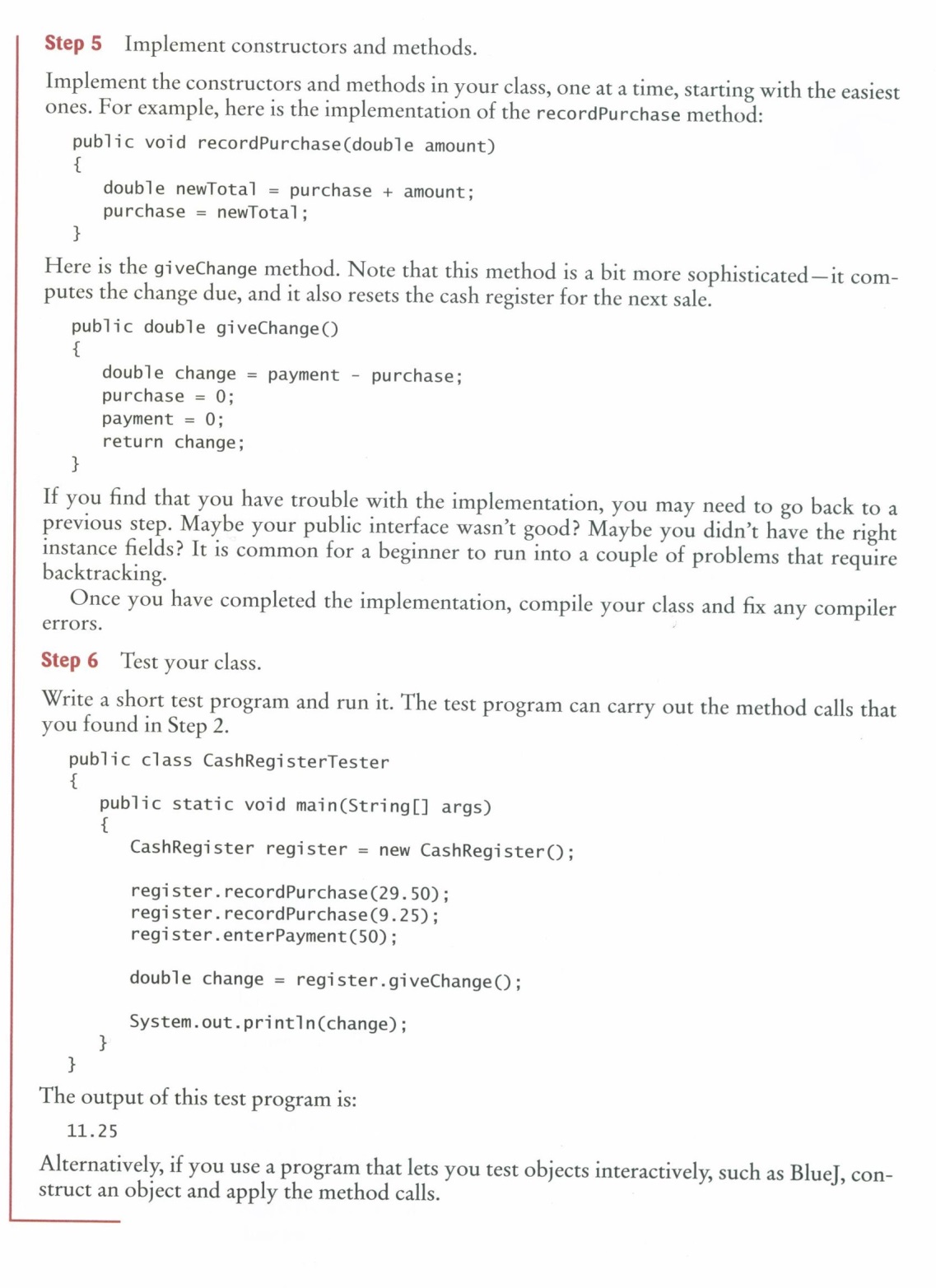
# WORKSHEET #8

## Assumes: Ch1, Ch2, Ch3, Ch5, Ch6, Ch7

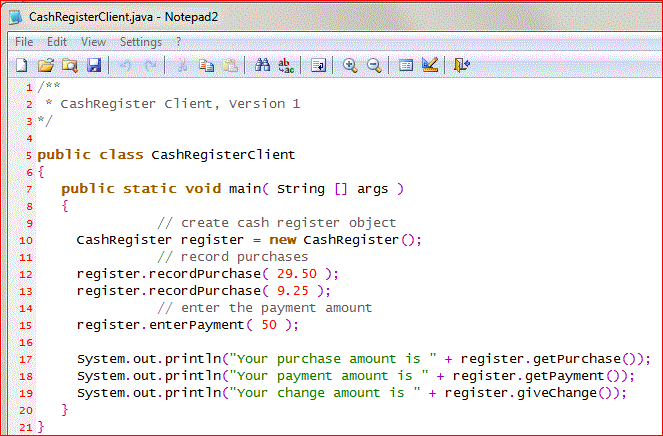
1. Researches have studied why some students have an easier time learning how to program than others. One important skill of successful programmers is the ability to implement the program **incrementally** and simulate the actions of a program with **pencil and paper**.

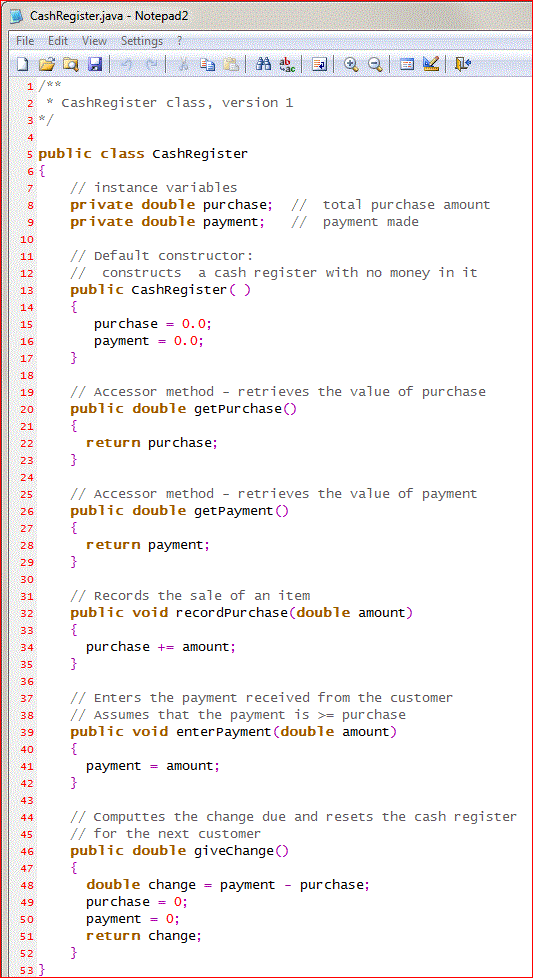






As the result we have created two java files: CashRegister.java and CashRegisterClient.java as shown below:





Your task is to use a graph paper and a pencil to trace this program.

Run this program and verify that it works as designed.

1. Enhance the CashRegister class so that it counts the purchased items. Provide getItemCount method that returns the count
2. Enhance the CashRegister class further:
   1. Add support for computing sales tax in the CashRegister class. The tax rate should be supplied when constructing a CashRegister object. Add recordTaxablePurchase and getTotalTax methods. Amounts added with recordPurchase method should remain not taxable. The giveChange method should correctly reflect the sales tax that is charged on taxable items.
   2. Add a method printReceipt to the CashRegister class. The method should print the prices of all purchased items and the total amount due
   3. After closing time, the store manager would like to know how much business was transacted during the day. Modify the CashRegister class to enable this functionality. Supply methods getSalesTotal and getSalesCount to get the total amount of all sales and the number of sales. Supply a method reset that resets any counters and totals so that the next day’s sales start from zero.
3. Write a program that analyses a given integer number. Create service class NumberAnalyzer.java and a client class NumberAnalyzerClient.java as follow:
   1. The service class has only one instance variable of type int called number.
   2. The service class has the following methods:
      1. a default constructor that generates the number randomly
      2. a secondary constructor that takes the number as an input passed from the client
      3. accessor, mutator, equals, and toString methods
      4. the following “business” methods:
         1. isMultipleof\_7\_11\_13() – returns **an int**: either 0 if the instance variable number is not a multiple of 7, 11, or 13; otherwise it returns the value of the multiplier. For example the method would return 0 if the number was 5; it would return 3 if the number was 21; it would return 10 if the number was 130
         2. isSumOfDigitsOddOrEven() – returns **a String**: “ODD” if the sum of digits in the instance variable number is odd; otherwise it returns “EVEN”
         3. isNumberAPrime() – returns **a boolean**: true if the instance variable number is a prime number; otherwise it returns false
         4. evaluate() – evaluates the instance variable number and returns **a char**: either ‘-‘, ‘+’ or ‘0’
         5. squareRoot() – returns **a double** representing a square root of the instance variable number, calculated by the following algorithm: the square root of a number N can be approximated by repeated calculation using the formula: NG = 0.5 (LG + N/LG); where NG stands for next guess and LG stands for last guess. N will be the starting value of LG. The method will compute a value of NG using the formula given. The difference between NG and LG is checked to see whether these two guesses are almost identical. If they are, NG is accepted as the square root; otherwise the new guess (NG) becomes the last guess (LG) and the process is repeated (another value is computed for NG, the difference is checked, and so on). The loop should be repeated until the difference is less than 0.005
   3. Client class has main and calls all your methods, try the following values for the numbers: 4, 120.5, 88, 36.01, 10000, and 0.25
   4. Draw a UML diagram for the service class
4. Implement a service class TimeDepositAccount and a client class to test it. Draw a UML diagram first. A time deposit account has a fixed interest rate that should be set in the constructor, together with the initial balance. Provide:
   1. secondary constructor,
   2. a method to get the current balance,
   3. a method to add the earned interest to the account – this method should have no arguments because the interest rate is already known to the object, and no return value because you already provided a method for obtaining the current balance
   4. toString and equals methods
   5. a withdraw method that removes the entire balance; partial withdraws are not allowed
   6. it should not be possible to deposit additional funds into this account
5. ☺