1. (Invoice Class) Create a class called Invoice that a hardware store might use to represent an invoice for an item sold at the store. An Invoice should include four pieces of information as instance variables—a part number (type String), a part description (type String), a quantity of the item being purchased (type int) and a price per item (double). Your class should have a constructor that initializes the four instance variables. Provide a set and a get method for each instance variable. In addition, provide a method named getInvoiceAmount that calculates the invoice amount (i.e., multiplies the quantity by the price per item), then returns the amount as a double value. If the quantity is not positive, it should be set to 0. If the price per item is not positive, it should be set to 0.0. Write a test app named InvoiceTest that demonstrates class Invoice’s capabilities.

2. (Employee Class) Create a class called Employee that includes three instance variables—a first name (type String), a last name (type String) and a monthly salary (double). Provide a constructor that initializes the three instance variables. Provide a set and a get method for each instance variable. If the monthly salary is not positive, do not set its value. Write a test app named EmployeeTest that demonstrates class Employee’s capabilities. Create two Employee objects and display each object’s yearly salary. Then give each Employee a 10% raise and display each Employee’s yearly salary again.

3. (Date Class) Create a class called Date that includes three instance variables—a month (type int), a day (type int) and a year (type int). Provide a constructor that initializes the three instance variables and assumes that the values provided are correct. Provide a set and a get method for each instance variable. Provide a method displayDate that displays the month, day and year separated by forward slashes (/). Write a test app named DateTest that demonstrates class Date’s capabilities.

4. Create a Dog class with three properties - name (String), age(int), weight(double). Override the toString() method using Eclipse's auto-generation feature. Create a test class to create two instances of Dog, set their properties, and then call the toString() on them. Add a showExcitement() method that displays the name of the dog + "is wagging its tail!!!" when invoked on the instance.

5. (Constructors) Use Eclipse to auto-generate a variety of constructor options (based off the Dog properties).

6. (overloading) Overload the Dog’s "showExcitement" method. Take in an int argument - levelOfExcitement - and customize the level of excitement displayed when the method is invoked. Also, create an eating() method that is overloaded. First method has no parameters and display “eating!” and the second, overloaded method takes in a String called food, and displays “eating (food)”

7.(encapsulation) Create a class called Person. Add three properties - name, age, shoeSize. Create a method called "walking" that prints out that the name of the person, their age, and with shoeSize is walking. Create another method called "startWalking," which will invoke the "walking" method created earlier. Encapsulate all properties and the "walking" method by setting their accessibility to private and creating getters/setters for each property. Create a driver class (with a standard main method) to use for testing the Person class.