Java Programming: Lab 7

OOP Inheritance and Static/Instance

In-class lab: Complete the Essentials problems in class. Paste your code into the worksheet, and answer any questions in italics. Save your worksheet and upload it to this lab's dropbox at the end of that class.

Next class, and in your own time: Start with the worksheet that you've been working on. Finish any incomplete Essentials problems.  Complete the Advanced problems online. Work on as many as you can. Again, paste your code into the worksheet and answer any questions in italics.  Save your worksheet and upload it to this lab's dropbox by the deadline - this is the day before we start the next topic in class.

You should comment all of your code!

You are encouraged to work with other students, and to give others assistance if they need. And, please ask the instructor if you have any questions.

Essentials – Work on these in class and upload at the end of class

Problem 1:

Create a class called InsectManager. This class should have a main module and will be used to create Insect objects, and test your Insect classes (and subclasses, which you’ll create in a moment).

And, create a class called Insect. This will represent a generic insect species. We’ll specialize it with subclasses soon.

For the Insect class, please add 3 variables:

A String to store the insect’s name

An int to store the number of wings

A constant to store the number of legs, and set it to 6.

Will these be public, private, or protected?

Add get and set methods for the first 2 variables. What will you do with the constant? Make it public, or add a get method?

Next, create another new class called Butterfly. Butterfly needs to extend Insect. Butterfly needs two methods

A String to store the butterfly’s wing color

A String to store the butterfly’s favorite flower

Will these be public, private, or protected? Add get and set methods for these variables.

Also, please add a constructor that takes 4 arguments – the butterfly’s name, wing color, number of wings, and favorite flower.

Also, add a method to the Butterfly class called printSpeciesData. This method should print all the info for a Butterfly object – it’s name, wing color, number of wings, number of legs, and favorite flower.

Test your Butterfly class by writing code in the InsectManager class to create some test Butterfly object. Create two Butterfly objects, and then call the printButterflyData method on each to display all the data for each of the Butterfly objects.

Suggestions: A Monarch butterfly has 6 legs, 4 wings, it is orange and black, and likes a plant called milkweed.

A Common Yellow Swallowtail butterfly has 6 legs, 4 wings, it is yellow and black, and likes a plant called milk parsley.

Next, create another subclass of Insect. This class should be called Bee. A Bee is a Bee, and a Bee is also an Insect.

Bee needs to have three variables:

A String for the bee’s color

A boolean for whether this species of bee makes honey (not all bees do)

A String for this bee’s favorite flower

Bee also needs a constructor to set all the variables a Bee object has. And, bee needs a method calledprintSpeciesData() that prints out all of the data for a Bee species. This method has the same name as the Butterfly method, but prints out the Bee information. As it prints out the information, it should print “This bee does make honey” or “this bee does not make honey” instead of “true” or “false”.

Test your Bee class by writing code in the InsectManager class to create some test Bee object. Create two Bee objects, and then call the printSpeciesData method on each to display all the data for each of the Bee objects.

Suggestions: name = “Honey bee”, color = “yellow and black”, makes honey = true, favorite flower = “Jasmine”, number of wings = 4

Name= “Bumble bee”, color = “yellow, black and white”, makes honey = False, favorite flower = “clover”, number of wings = 4

Last task: Create a LinkedList of Insects (recommended: use Generic types to require that the list can onlycontain Insect objects).

Use a for each loop to iterate over your list and call printSpeciesData for each Insect.

LinkedList insects = new LinkedList();

//Create and add some Butterfly and Bee objects to the list

for (Insect i : insects) {

i.printSpeciesData();

}

Unfortunately, this doesn’t work. The insect class does not have a method called printSpeciesData. While Bee and Butterfly both do, there is nothing to guarantee you won’t add an Insect object to this LinkedList, and Insect does not have this method. So the compiler complains.

To fix, please add this code to Insect

public abstract void printSpeciesData();

And change the first line of the class from

public class Insect { //class definition follows

To

public abstract class Insect { //your class definition here

Now your code should work.

“Abstract”, for the method, means that all Insect objects will have a method called printSpeciesData. But Insect doesn’t want to define exactly what printSpeciesData does, and Insect doesn’t need to implement it. This is useful if you will never instantiate your superclass, if it doesn’t make sense to make objects from the superclass. In this program, it doesn’t make sense to make Insect objects, it’s only useful to make the Butterfly and Bee subclasses. But, you want every insect to have printSpeciesData. So declare it as Abstract in Insect, and the compiler will make sure all subclasses of Insect will create a printSpeciesData method.

You also need to change the class to an Abstract class. You can never make objects from an abstract class. After all, you haven’t written all the method definitions it needs.

This is useful in any scenario where you have a general description class, and then specialize it with many subclasses; and only the subclasses will be useful in your program. You can now specify more required behavior in the (abstract) superclass and enforce that the subclasses will be able to have this behavior too... helps with polymorphism.

Notice that printSpeciesData should call the method of that name for the correct type of insect – this is polymorphism! So it should call printSpeciesData from the Butterfly class if the Insect object in the list is a Butterfly, and it will call the printSpeciesData from the Bee class if the Insect object in the list is a Bee.

This example uses generic types to make the LinkedList only able to store Insect objects. Please see the Data Types lab for more information and examples on generic types.

Please paste your Java code here:

Problem 2:

Fork and pull the HVAC service call program from

HYPERLINK "https://github.com/minneapolis-edu/HVAC" https://github.com/minneapolis-edu/ HYPERLINK "https://github.com/minneapolis-edu/HVAC" HVAC

Review the Furnace class. There are a lot of static things in this class.

Question: Identify all of the static things, and the reason why each is static.

Problem 3:

In your HVAC program, create a new class called WaterHeater. This represents a service call for a water heater. A water heater service call needs an address, date service requested, description of the problem, and the age of the water heater. Resolved service calls also need the resolved date, description of the resolution, and the fee charged to the customer.

The city requires that all service calls to water heaters have a mandatory $20 extra charge added. As this applies to all water heaters, add a static variable to your class to store this data.

Add a toString method to WaterHeater which returns a string containing all the static and instance variables for a WaterHeater. You should break down the fee into the service charge plus the $20 mandatory city fee.

Add code to HVAC.java to test your new class. Make sure you can add service calls for water heaters to the list of todayServiceCalls.

Please paste your java code for HVAC.java and WaterHeater.java here

Advanced

Finish any incomplete Essentials problems first.

All of these questions refer to the support ticket application from Week 7 – More OOP – Support Ticket Program powerpoint slides. Please review these slides and get the code running.

Run and test with some example support tickets.

Answer the questions from Problem 1 and Problem 5. For your code, when you have finished all the problems, paste in your code at the end AND push it to your GitHub repository.

Problem 1:

In your own words, Explain the role of the static and instance ticketID variables in the Ticket class. Why are a static variable, and an instance variable, declared in the class?

Problem 2:

Add code to the deleteTicket method; so if user enters a ticket ID that doesn’t exist, it prints an error message and asks the user to try again.

You should also validate that the user is not entering a String or double or something that is not an int. Again; the method should ask the user to enter the ticket ID again.)

Problem 3:

Modify the menu. Remove the Delete Ticket option. Replace it with 3 new options: Delete by ID, Delete by Issue, and Search by Name.

Write ONE method, which searches your ticket list and returns a new list of Tickets whose descriptions contain a certain string. For example, you might want to search for all tickets with the word “server” in. Your method should return a list of all Tickets with “server” in the description.

Use this method to help you implement Search by Name.

Problem 4:

Now, implement Delete by Issue. Your user may want to search for all tickets with "Server" in the description, to see a list of those tickets and their IDs. At that point, they could enter an ID of the Ticket they want to delete.

Problem 5:

Modify your program so that Tickets can store another date – resolution date – and a String that documents why the ticket was closed – the fix or the resolution for the ticket.

Now assume that when users delete a ticket, it has been resolved in some way. Either you’ve fixed the problem, or the user has figured out how to change their own screensaver, or it’s become a non-issue in some other way.

Now, when you delete a Ticket, your program should ask the user for the resolution. It should store the resolution, plus the current date. Now, remove this Ticket from the ticketQueue list.

And, add this ticket to a new list, called resolvedTickets.

There are at least two ways of doing this:

Question: Would you rather subclass Ticket and create a new class called ResolvedTicket? Or modify the current Ticket class to add these two new variables? Why did you choose the approach that you used?

Problem 6: When the program closes, please write out all the data about all open tickets, and all resolved tickets, to files.

Resolved tickets should go into one file. This file should have today’s date in the filename. Something like “Resolved\_tickets\_as\_of\_february\_14\_2014.txt” perhaps?

Open tickets should go in another file called “open\_tickets.txt”.

Problem 7: When you program opens, it should look for a file called open\_tickets.txt. Read in this file, and create ticket objects, and store these in the ticketQueue list so the user can see all open tickets.

Problem 8: What happens to ticket IDs when the program is closed and opened? Make sure they don't reset to 1 when the user restarts the program.

Push this Java code to your GitHub repository and paste the link here

Also paste your Java code here